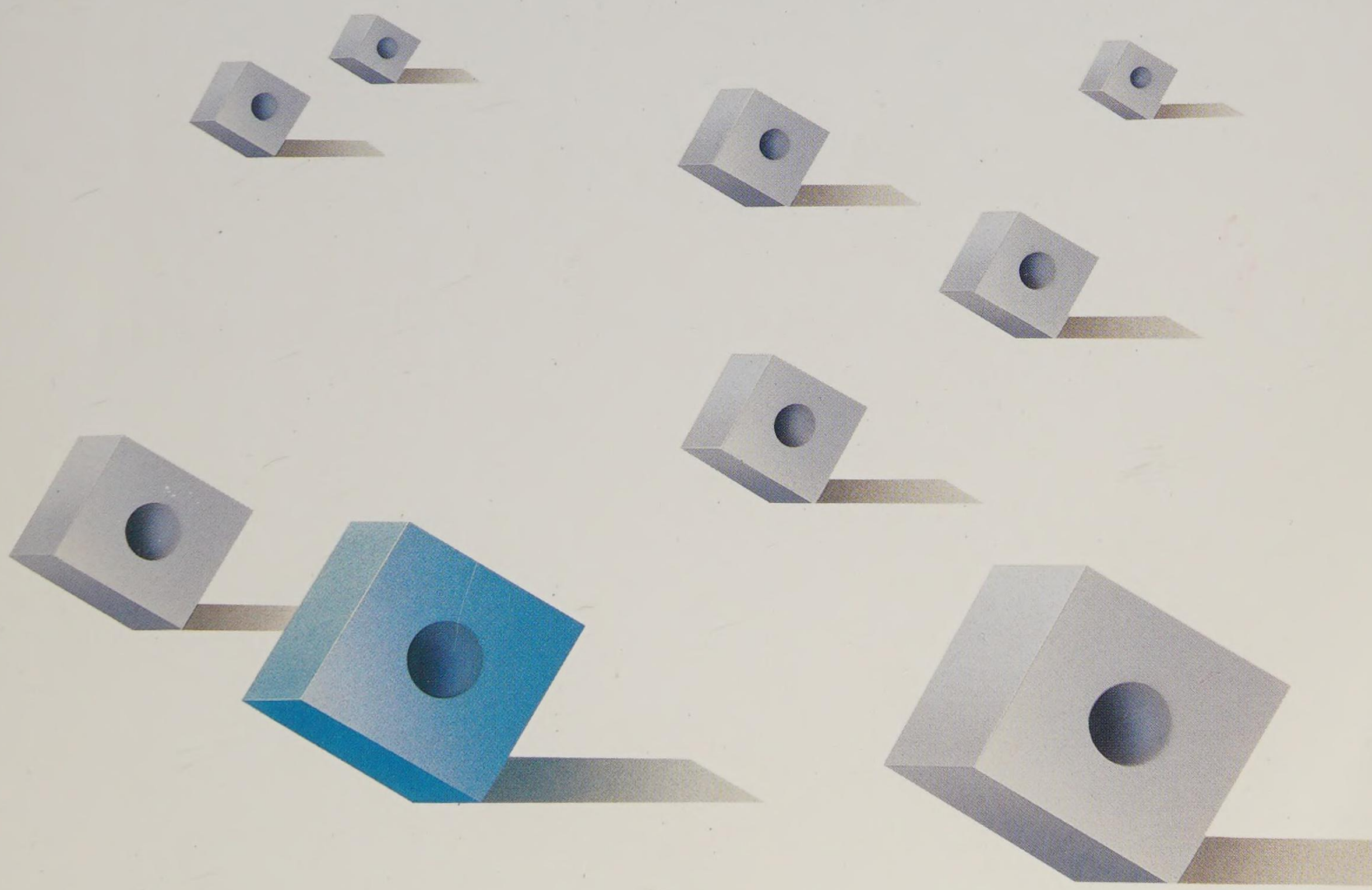
WINDCREST®/McGRAW-HILL

DOS 5 Demystified

James S. Forney



▶ Covers all versions through 5

DOS 5 Demystified

To Shiela

DOS 5 Demystified

James S. Forney

WINDCREST®

FIRST EDITION
FIRST PRINTING

© 1991 by **James S. Forney**.

Published by TAB Books.
TAB Books is a division of McGraw-Hill, Inc.

Printed in the United States of America. All rights reserved. The publisher takes no responsibility for the use of any of the materials or methods described in this book, nor for the products thereof.

Library of Congress Cataloging-in-Publication Data

Forney, James.
DOS 5 demystified / by James S. Forney.
p. cm.
Includes index.
ISBN 0-8306-1048-0 (h) ISBN 0-8306-1047-2 (p)
1. Operating systems (Computers) 2. PC-DOS 3. MS-DOS.
I. Title. II. Title: DOS five mystified.
QA76.76.063F642 1991 91-20914
005.4′46—dc20 CIP

TAB Books offers software for sale. For information and a catalog, please contact TAB Software Department, Blue Ridge Summit, PA 17294-0850.

Acquisitions Editor: Stephen Moore
Technical Editor: Kellie Hagan
Production: Katherine G. Brown
Series Design: Jaclyn J. Boone WT1

Contents

Acknowledgments *xi*
Preface *xiii*
Introduction *xv*

PART 1. DOS BASICS

Chapter 1. What is DOS? 3
Help wanted *5*
Bring in the interpreter *7*
What? Not COMMAND.COM even? *7*
Internal vs. external *8*
On the outside *9*
Straight to the heart of DOS *10*

Chapter 2. Stripped to the bare essentials 11
Basic I/O services *12*
Enter DOS *13*
The loader *15*
Beyond just basic I/O services *16*
The DOS kernel *17*

Chapter 3. The final puzzle piece 21
The pecking order *22*
Even the road to "you know where"
is paved with good intentions *23*
Batch file processing *24*
The great—or not so great—communicator *25*

PART 2. INSTALLING AND CONFIGURING DOS 5

Chapter 4. Installing DOS 5 29

Upgrading from an earlier DOS version *30*
Lovers and other strangers *31*
Go for it *32*
Let SETUP do it for you! *32*
The CONFIG.SYS file *33*
Cleaning up *35*
Installing 5.0 on a new machine *36*
For use on networks *37*
Reaching for new highs *37*
Decisions, decisions *38*
Upper memory blocks: memory DOS has never given you before *38*

Chapter 5. Doing DOS my way . . . or yours 39

The CONFIG.SYS *40*
The AUTOEXEC.BAT *52*

Chapter 6. The environment 55

COMSPEC *56*
PATH *58*
PROMPT *60*
SET *62*
The odd couple *65*
Parent and child *67*

PART 3. PRODUCTIVITY

Chapter 7. Help, help!! 71

Chapter 8. Fact and fiction 77

Shell game *78*
A brave new world for DOS *79*
Speed demons *80*
The wrong way around with APPEND *85*
SUBST: the shortest route *85*
Batch file bingo *86*
Past their prime *86*

Chapter 9. Batch files and DOSKEY macros 87
At the most elemental level *88*
A language of their own *89*
Symbols with special meanings *90*
Nothing has no meaning *92*
Great one-liners *93*
Exceptions to almost any rule *94*
The same name, but different *95*
The big *IF* *96*
Macro or batch file? *99*
Other considerations *102*

Chapter 10. Memory management: above and beyond 103
Grass roots support *105*
Burning the candle at both ends *106*
First come first served *107*
HIMEM.SYS, the easy part *108*
Only for the upper class *109*
Grabbing unused memory *110*

Chapter 11. Excess baggage 115

PART 4. DISK MANAGEMENT

Chapter 12. The new order 123
Disk media: order out of chaos *123*
Disk space: lost and found *124*
Fragmentation: chaos out of order *128*

Chapter 13. Directories of one kind or another 135
Attribute bits *137*
The eloquent tree *138*
New kid on the block *139*
Around and around *141*

Chapter 14. Copycats 143
Beyond the old familiar *144*
The archive bit *145*
Up that down tree again *147*
Replacing what was never there *149*
Safety in numbers *149*
Less-than-perfect security *151*

Chapter 15. Close encounters and disasters 153
The simple "OOOPS!" *154*
Tracking and recovering erased files *154*
A kinder, gentler format *157*
Mirror, mirror, on the wall . . . *159*
The ultimate test *161*
Not really what it seems *162*
More than the proverbial ounce *163*
Saving system information *164*
Saving hard disk partition data *166*
What does it all mean? *167*

PART 5. BEYOND JUST DOS

Chapter 16. The new DOS shell 171
Modes and resolutions *173*
File management *174*
Head 'em up and move 'em out *175*
The main event *176*
Beyond the basic defaults *178*
Task switching *180*
Another near miss *183*

Chapter 17. Goodbye EDLIN, hello EDIT! 185
Don't confuse it with word processing *186*
A touch of the familiar *187*

Chapter 18. QBASIC: more than just a pretty face 191
The more things change . . . *192*
More help *193*
Expert systems *196*
Converting other BASIC programs *196*

Chapter 19. DEBUG: don't let it bug you 199
Everybody's doing it *200*
Doing the unspeakable with DEBUG *201*
Scripts and more scripts *204*
Don't let it bug you *204*

PART 6. THE WHOLE NINE YARDS

Chapter 20. Everything that's fit to print 209

ANSI.SYS *212*
APPEND *216*
ASSIGN *220*
ATTRIB *222*
BACKUP *225*
BREAK *229*
BUFFERS *231*
CALL *233*
CHCP *234*
CHDIR (or CD) *235*
CHKDSK *237*
COMMAND *240*
COMP *242*
COPY *244*
COUNTRY.SYS *246*
CTTY *247*
DATE *249*
DEBUG *251*
DEL (ERASE) *253*
DELOLDOS *255*
DEVICE *257*
DEVICEHIGH *258*
DIR *259*
DIRCMD *262*
DISKCOMP *263*
DISKCOPY *264*
DISPLAY.SYS *267*
DOS *268*
DOSKEY *269*
DOSSHELL *273*
DOSSWAP *274*
DRIVER.SYS *275*
DRIVPARM *277*
ECHO *279*
EDIT *281*
EDLIN *282*
EGA.SYS *283*
EMM386 *284*
EXE2BIN *286*
EXIT *287*
EXPAND *288*
FASTOPEN *290*
FC *291*
FCBS *293*
FDISK *294*
FILES *296*
FIND *297*
FOR *299*
FORMAT *301*
GOTO *305*
GRAFTABL *307*
GRAPHICS *308*
HELP *310*
HIMEM.SYS *313*
IF *315*
INSTALL *317*
IO.SYS *318*
JOIN *320*
KEYB [.COM] *321*
LABEL *322*
LASTDRIVE *323*
LOADFIX *324*
LOADHIGH *325*
MEM *327*
MIRROR *331*
MKDIR (or MD) *335*
MODE *337*
MORE *345*
MSDOS.SYS *347*
NLSFUNC *348*
PATH *349*
PAUSE *351*
PRINT *353*
PRINTER.SYS *357*
PROMPT *358*
QBASIC *360*
RAMDRIVE.SYS *361*
RECOVER *363*
REM *366*
RENAME (or REN) *368*

REPLACE *369*
RESTORE *371*
RMDIR (or RD) *373*
SET *375*
SETVER *377*
SHARE[.EXE] *380*
SHELL *380*
SHIFT *383*
SMARTDRV.SYS *385*
SORT *387*
STACKS *389*
SUBST *390*
SWITCHES *392*
SYS *393*
TIME *394*
TREE *395*
TRUENAME *397*
TYPE *398*
UNDELETE *400*
UNFORMAT *402*
VER *405*
VERIFY *406*
VOL *407*
XCOPY *408*

Index 411

Acknowledgments

Special thanks to Tanya Van Dam and Cristy Gersich in Corporate Communications at Microsoft for seeing to it that I always had the tools to work with close at hand. Thanks also to Eric Straub and Eric Hough for their technical expertise and guidance through some of the more complex intricacies of DOS 5.

Thanks finally to my many friends who, though we were all under nondisclosure, managed to convey to me their satisfaction with the product as it was emerging, reinforcing my own confidence in this new release.

Preface

One of the biggest problems in writing this book was ongoing changes in the product during the roughly year-long beta program. During that time, various new functions appeared, sometimes mysteriously and often undocumented. Some of them disappeared as quietly as they had appeared. Or, as in the case of one pair, originally REBUILD and UNFORMAT, the functions were merged into a single new UNFORMAT utility.

In the early stages of the beta program, there was no support for certain memory management features and it appeared, despite some hype, that this release would offer little if anything more than the rather rudimentary memory manager supplied with Windows 3.0. There was no support for mapping memory to upper memory blocks on 80386s at all—an area of special interest to me. But as things progressed, a far more powerful HIMEM.SYS and its companion EMM386.EXE evolved to form a really credible new duo.

Some new switches were still being added to old commands right up to the final pre-release, and at least one completely new utility was added in the final pre-release. Because of the fluid nature of the situation, official documentation was often unavailable (though one or more members of the 5.0 team at Microsoft were most generous of their time and, without their help, this book could not have been completed).

Throughout the process, I made every effort to check and recheck findings and results with every new release to make this book as accurate as possible. During the writing of this book I worked exclusively with 5.0 on a pair of 80386s, my wife's souped-up 80286, a couple of old 8088s, and even on my laptop when traveling.

Still, despite all good intentions, it's not only possible but probable that errors have slipped through. It's even possible that the final release will differ slightly from the "final" version I had in hand when this book went to press.

Changes in programs are an ongoing process in this business, even in releases already on the market.

Also, keep in mind that this book is not intended to replace the official documentation—rather to supplement it from a user point of view, concentrating on the issues I think important to the greatest number of users and, with the exception of the final section where I've examined everything in 5.0 in a pseudo-documentation format, skipping over lesser areas.

In any event, this book is the result of the most exhaustive project I've ever undertaken. I've tried to boil all the good stuff down so that, even if some of the minor details change, you'll still get the general sense of what things are and how they work.

I've tried to look at DOS 5.0 fairly, but it's a fairness tempered by experience with many excellent 3rd-party packages. I've tried to point out where DOS 5.0 is strong and where it isn't. Still, it's a good release—a truly great release as I look back, especially, at 4.0. I can only hope this book will help bring to you the enjoyment this new release has brought to me.

Introduction

This is certainly the best, the most powerful MS-DOS I've seen to date—a one-size-fits-all DOS—that as an upgrade option for the first time is the exact same item for genuine IBM machines, clones, and semi-clones alike.

This is a really good DOS, but it isn't perfect. Nothing ever is. I have, with the exception of occasional comparisons, confined myself primarily to what comes standard with the DOS 5.0 retail upgrade kit package, and tried to point out DOS's shortcomings as well as its virtues. And, to keep the title honest, I've tried to strip away the mystery, the *mystique* of DOS and make it understandable in ways I hope will make it easier and sometimes even fun, perhaps, to work with.

In the wake of the DOS 4.0 fiasco that sent many of the relatively few who even tried it fleeing back to version 3.*x*, there was a lot of speculation as to DOS's future . . . if, in fact, it had a future. There was no OS/2 or other really viable alternative waiting in the wings for such an opportunity. And so we waited, left to ponder.

Well, wait no longer! DOS 5 is here and it's all the things you might have wished for 4.0 and 4.01, and more. Bruised by the tarnished image left by 4.0, Microsoft has really gone the extra mile this time, not only adding features that should keep DOS viable, but making sure that it's still the operating system of choice for some years to come.

What's new in DOS

This new release is packed with features you've been waiting for, including on-screen help. And, for the first time, Microsoft is even offering direct support via a new 900 number. Among other new DOS features, 5.0 has a brand-new nondestructive FORMAT and an UNFORMAT in case you goof, as well as an UNDELETE. DOS 5 even has a painless way to recover from most hard disk crashes.

The shell, first seen in 4.0 but far more powerful in 5.0, has added task switching. GWBASIC has been replaced by QBASIC, a slick new menu-driven

interpreter that, other than not being a compiler, now does almost everything QuickBASIC does. And EDLIN . . . poor old EDLIN. It's still there, but 5.0 also has a brand-new menu-driven EDIT feature that's really worth its weight. And there's more, much more—and yet, for all that's new, you can slip right into using 5.0 without relearning anything, because the old commands and syntax all still work. But you're going to want to stop with what you know already when you see all the things DOS 5.0 can do for you.

While fully compatible with 8088 machines, for the 286 and higher set, DOS 5.0 can move most of itself out of conventional memory to give you at least an extra 47K of workspace for your applications—overall, up to perhaps an extra 200K or so on 386 and higher systems. And not surprisingly, DOS 5.0 is tailor-made, not only just to run Windows 3.*x*, but to enhance the Windows environment. Windows—like so much other software—can no longer really live in just 640K, so DOS 5.0 goes where no DOS has gone before to support the almost unlimited memory potential of today's most powerful machines.

Something for everyone

DIR Greatly enhanced in 5.0 to include about a dozen sort options, plus another ten based on attributes (including switches for displays limited to or excluding directories). Any of these options can be set as defaults or invoked from the command line on a one-shot basis.

DOSKEY New TSR utility that allows for command line editing and recalling and editing of prior commands (to the limit of the stack), and also provides support for new macro functions roughly paralleling many batch file uses.

DOSSHELL Significantly enhanced from the shell first seen with DOS 4.0, this new feature includes an elementary context-switching task swapper. Primarily a menu-driven file management tool with context-sensitive help, DOSSHELL is designed with the mouse user in mind.

EDIT After years of kicking poor old EDLIN around, Microsoft has finally come up with a slick new full-screen text editor this time—EDIT.COM. Powerful enough for many simple word processing tasks like writing memos or quick correspondence, it's also a good tool for creating or editing CONFIG.SYS, AUTOEXEC.BAT, and other batch files.

HELP After years of cries for help falling on deaf ears, Microsoft has gone overboard this time by offering on-screen help for command-line functions, as well as special help facilities for the new DOSSHELL, QBASIC, and EDIT features. DOS 5 also provides an extensive help facility for the QBASIC interpreter.

QBASIC A glossy new user-friendly BASIC interpreter with on-screen help, which completely replaces GWBASIC (and, when upgrading IBM machines,

replaces BASICA). Based on Microsoft's powerful QuickBASIC compiler, this one does almost everything except compile.

A number of disk management features

FORMAT This new formatter has a nondestructive default mode for floppies as well as hard disks. Based on technology licensed from Central Point Software (PC Tools), this is the cornerstone of a powerful new disk management scheme that includes an interrelated set of unformat, undelete, and disk recovery utilities.

MIRROR Also based on PC Tools-licensed utility, MIRROR offers multiple levels of disk and data protection, including backup of the FAT, root directory, and hard disk partition table. It also provides a special tracking file for tracking and undeleting erased files (providing that the data space hasn't been overwritten).

UNDELETE A new multilevel undelete utility that provides full file recovery—including full filename when the delete tracker function of the MIRROR program is used. UNDELETE can also unerase files when the delete tracker hasn't been run, but the first letter of filenames must be restored manually.

UNFORMAT Far more powerful than the name implies, this new utility can also restore the FAT, root directory, and hard disk partition table in the majority of hard disk crashes, if the companion MIRROR program has been run as recommended. It can also fully restore accidentally formatted hard or floppy disks.

In addition, there is hard disk support for disk partitions to 2 gigabytes in size and for more than two hard disks on a system, and floppy disk support for the new 2.88Mb 3 1/2-inch drives.

Where DOS has never gone before

HIMEM.SYS A new XMS memory manager for 80286 and higher systems. It's the key to new DOS 5.0 memory management scheme for use with any computer having extended memory capability.

DOS=HIGH Enables DOS to load 47K of itself to the High Memory Area (1024K to 1088K) on 286 and higher machines when HIMEM.SYS (or other XMS driver) is installed and extended memory is available. It leaves only 17K of DOS in low memory, freeing the remainder for use by applications.

DOS=UMB Enables memory mapping to Upper Memory Blocks—unused address spaces between 640K and 1024K—when the EMM386.EXE EMS emulator is installed in addition to HIMEM.SYS (for 80386 and higher systems only).

EMM386.EXE New DOS EMS emulator for use with 30386 and i486 systems. HIMEM.SYS or other suitable XMS driver must be installed. Provides for all LIM 4.0 EMS services.

DEVICEHIGH Used in CONFIG.SYS instead of the DEVICE command to force device drivers to load in UMBs if upper memory is available and they aren't too large to fit the space available.

LOADHIGH Used from the command line (or batch file) to force TSRs to attempt to load in Upper Memory Blocks, if available and if the program isn't too large to fit the space available.

MEM Introduced with 4.0, but significantly enhanced in this release to provide analysis of available memory resources, memory usage, etc. MEM is especially valuable when working Upper Memory Blocks (640K to 1024K).

Special services

INSTALL A completely new, virtually foolproof installation program that even saves the entire DOS you're currently using plus your configuration files so you can restore it intact if you need to.

LOADFIX With DOS 5.0 now able to load so much of itself plus most device drivers above 640K on 386 and higher systems, and having a lot of additional conventional memory to work with, some programs have been found to have a problem loading, particularly when DOS tries to load them to addresses below 64K. To cope with this problem, DOS 5.0 adds LOADFIX to its list of utilities.

SETVER Good as this release is, incompatibilities with some existing software are as sure as death and taxes. So in version 5, Microsoft has added SETVER to 5.0 to reduce the impact of incompatibilities by fooling some programs into thinking they're actually running under earlier versions of DOS.

Network redirectors Provided with the basic ugrade kit for several of the most popular networks. (Many more are available in the optional extra network support package.)

New direct technical support Although poorly documented in the documentation available from Microsoft at this writing, Microsoft will provide free—yes, free—technical support for customers buying the retail upgrade package. (Although many users probably aren't aware of it, beginning in April of 1990 Microsoft began offering $2-per-minute technical support for earlier DOS releases to anyone calling 1-900-896-9000.) Check with whatever vendor you purchase your upgrade kit from for further details.

The compatibility issue

DOS 5.0 clearly represents the most significant change not only in DOS itself, but Microsoft philosophy since 3.0 in many ways. Of course, you can't simply take what's new in 5.0 out of context and devote a book to that. So, especially for users

new (or newer) to DOS, I'll look at some of the more significant areas in general terms.

Because so much *is* new in 5.0—brand-new in many cases, significantly enhanced in others—it's important to point out up front that the casual user will see little if any change. This is important because it means that upgrading to benefit from 5's new features doesn't mean relearning—or, in the business world, having to retrain everyone who's going to use it.

Through the years, Microsoft has painstakingly maintained something called *backward compatibility*. The term is usually construed to mean that successive new releases will, although geared to newer generations of machines, peripherals, and user needs, still be compatible even with those early 8088 PCs still out there, with only rare exceptions.

More important, though, is the backward compatibility DOS has maintained in its command syntax even as that syntax has, in many cases, been supplanted by greatly enhanced command sets and switches. If, for instance, under 3.x you had to tell FORMAT how many tracks (/t:*nnn*) and sectors (/s:*n*) per track it had to format on upgraded disk drives, those same switches and numbers—if you can remember them—still work with version 5.0. They still work even though DOS 4.0 introduced an all-new set of logical and easy-to-remember switches that would do the job instead.

All the old switches and numbers still work with 5.0, even though in this release FORMAT.COM has been totally rewritten to do a nondestructive format that allows for the total recovery of all files and data after an untimely reformatting. And they've maintained compatibility and continuity by continuing to support outmoded utilities like EDLIN, even as DOS introduces a slick, new, easy-to-use, full-screen text editor called EDIT in this release.

So, when looking at the new DOS, do look carefully—not at the old stuff that's still in there, but at what's new. And keep in mind that, in many cases, the new lies behind the seemingly familiar. It's a new day and this is a new DOS geared to a new generation of desktop computers with speed, power, and memory capabilities far beyond even the wildest dreams of all the reckless visionaries just a few years back. It's a new DOS, the best I've seen from Microsoft yet, coming like a breath of fresh air after 4.0 and 4.01—and the less said about *them* the better.

Do I need to upgrade?

Now for the sixty-four dollar question, or the $99.00 question, or whatever tag the dealer puts on it—"Do *I* need to upgrade to DOS 5.0?" Much of the answer to that question depends on:

- What version of the operating system you're using now
- How fully you're currently using what you have
- Your needs for the foreseeable future

If, for instance, you're still using DOS 2.0, then it's definitely time—past time, even. Even if you're still running on an 8088 and have only limited expanded memory needs that are adequately met by what you're using now.

If you're one of the many who tried DOS 4.0 and went back to 3.3 or something earlier, it's almost certainly time to upgrade now. Even if the only thing new to this release of interest to you is the new disk management collection, it's time. If MIRROR pulls your FAT out of the fire just once, then the time, the effort, and in many cases the actual money that this can save you could repay the cost of upgrading many times over. In fact, for my money I think that almost any serious DOS user should upgrade to 5.0. It's *that* good.

So this is serious DOS, still compatible with all the 8088 machines that are still out there doing yeoman service, but at the same time geared to meet the challenge of the fast-track world of 80386 and 486 machines. This is DOS 5.0 viewed not just in the context of what it does, but in the context of the world around it—not just a world of Windows, but a world of opening doors.

Part 1

DOS basics

For all that's changed in this release, the real power behind DOS 5, as always, comes down to just two files, the hidden system files. All that DOS 5 can ever do is provided for in those two files—which is why it's important to update to this release as more sophisticated software looks for services that weren't provided earlier.

Of course, you must also have a means of accessing the system, a command interpreter, but the central core of DOS —the real DOS—is much smaller than most users realize.

CHAPTER 1

What is DOS?

Before you jump into DOS 5.0 with both feet, I really need to clarify just what DOS is—and what it isn't. We all know that DOS is an operating system, but I find that most users are really pretty fuzzy about just what that really means.

Simply put, an *operating system*—any operating system—is simply a set of instructions that tells a computer how to perform elementary tasks. Does this mean word processing, recalculating a spreadsheet, or sorting a database? No. Those functions also involve sets of instructions, *programs* that enable you to conduct some very specific business with your computers. At the operating system level, we're talking basics—like how to write to the screen or read from or write to some kind of storage media.

By itself, an operating system can do nothing more than create a working environment in which you can operate. If the operating system doesn't provide—or is incapable of providing—the basic services required to perform some complex function, it's unlikely that that function can ever be accomplished under the operating system. Or at least under that *version* of that operating system.

This is why software packages typically require DOS 2.1 or higher, in some cases 3.1, or some other specific floor. DOS 5.0 is a more powerful platform than DOS 4.x—which, at least in theory, was more powerful than DOS 3.x, and so forth. Each successive release has managed to squeeze a little more performance—or at least the potential for a little higher level of performance—out of the CPU chip, in DOS's case, the Intel 8086 family of computer chips.

The operating system for most computers today, however, is not a single package marked "operating system," but rather several layers—at least two layers, but in today's world often more. DOS constitutes one layer. It's the foundation on which users build. It isn't, however, the foundation on which the machine itself builds. There is a level under DOS, and it's at this level that the designer who

created your machine endowed it with a basic set of functions, typically the ability to:

- Self-test its various essential internal functions and test for the presence and functionality of default input/output devices, like the keyboard.
- Operate the default block device—hard or floppy drive—and enable it to read data (further instructions) into memory.
- Display at least a limited set of error messages, such as *Non-system disk . . .* or other error condition as appropriate.
- Read keyboard input and, at least to some extent, interpret or respond to certain keystrokes—any key to retry disk read, Ctrl – Alt – Del to restart the boot cycle, etc.

This is sketchy rundown of just some of the most basic operating functions that must be carried out before DOS, UNIX, or any other operating system even enters the picture. The need to self-test is self evident. But when you stop to think about it, so are the others I've listed here. If the brains a machine was born with, so to speak, aren't enough to run a disk drive or some other acceptable input device—at least enough to read DOS into memory—you don't have DOS. And, of course, if the machine can't read those further operating instructions for some reason, it really needs some way to communicate that fact to you.

These are primarily basic input and output functions—press a key, that's input; see the letter that it represents on screen, that's output. So that's what we call it—*Basic Input/Output Services*, or *BIOS* for short. The instructions that tell a barely wakened computer how to perform these basic functions are on a special memory chip that, once programmed, can be read and reread indefinitely, but not rewritten—therefore, it's called *Read Only Memory*, or *ROM* for short. Put it all together and you've got something called the *ROM BIOS*, a mini operating system—*the* operating system at the lowest level. In many cases, the original ROM BIOS can be replaced with an updated version, but this is the only way the ROM BIOS can be altered.

I'm talking about pretty rudimentary communication here, but it's interactive—which is all any computer really is, anyway. So even without DOS, that box of electronic goodies is a primitive computer of sorts. In fact, that's all the operating system for the first PCs from IBM anticipated: basic input and output services for a keyboard, and display. And for "mass storage," the ROM BIOS provided interfacing to an external cassette tape recorder—really sophisticated stuff, but then nobody, IBM especially, thought the PC was ever going to be more than just a toy in those days.

Software for PCs—software as we know it anyway—was nonexistent at that time, so IBM threw in a crude form of BASIC—real fun stuff—to play with. That too was ROM-based, because even floppy drives were thought to be too expensive for the limited market the PC would attract. In fact, even the brand-new PC-DOS

5.0 must look to some of the proprietary ROM routines unique to IBM machines, most notably part of the BASIC kernel.

Help wanted

Other than for the bare operating essentials endowed by the designers, your machine can't do a whole lot more for you without some outside help—without a lot of help, in fact, and the more complex the application software you want to run, the more help it needs. This is where DOS comes in. DOS picks up where those basic bodily-function type of instructions of the ROM BIOS leaves off.

DOS adds the functionality you need to do real work. In some cases, DOS actually overwrites (for the current session only) some of those basic low-level instructions with a more advanced set. And, last but not least, DOS provides an interface between you, your applications, and the underlying hardware.

As an interface, DOS functions at two quite different levels. At its lowest level, often referred to as the *kernel*, it's completely hidden. When you enter keyboard commands, you don't communicate them directly to the actual operating system, but rather with something called a *command interpreter*, in most instances COMMAND.COM. However, it's generally only at the more visible level of the command interpreter that you're ever involved with DOS, which unfortunately has led to a good deal of misunderstanding about just what DOS is.

The command interpreter, in fact, is really only just an interface to DOS—the real DOS. The real DOS is buried down there in two hidden files: MSDOS.SYS and IO.SYS, or IBMDOS.COM and IBMBIO.COM with PC DOS. (In the case of these two system files, DOS can and does use the .SYS and .COM extensions interchangeably.) It's those two hidden files that determine what DOS can and can't do, and are the *real* reason DOS must be updated periodically to keep pace, not only with new software, but today's more powerful machines.

Look at almost any software you're using, for example. Chances are that somewhere in the documentation—usually right up front—it will say something like "Requires DOS 2.0 or higher." And, while 2.0 might be good enough to satisfy your software, just try booting up with anything before 3.0, and the chances are that you can't even find your hard disk (something 2.0 didn't anticipate, but 3.0 took in stride). Before DOS incorporated hard disk support, it was necessary to install external device drivers from the CONFIG.SYS. This is still the case in any release, when you're dealing with devices that don't have built-in internal support.

As technology advances, each generation of new machines incorporates new features. 5.0, for example, already has support built into it for 2.88Mb 3½-inch floppy disks, something no one had ever heard of when DOS 4.0 was conceived. And for the first time DOS can move most of its own bulk of conventional memory and relocate just above 1024K on 286 (and higher) machines, something that in itself required a major rewrite.

On the other hand, in many cases even old machines can gain a whole new lease on life by simply going to a later DOS release. Later versions often allow you to access hardware capabilities that might have been there all the time but not been needed. DOS 5.0 offers a whole new level of memory support not seen in earlier releases—much of it completely meaningless before the introduction of the 80386.

The extent of change is particularly evident in IO.SYS in this release. In Fig. 1-1, for comparison, I've listed the more significant functions supported by IO.SYS in three representative DOS releases (not including COM and printer ports, etc.). Seen side by side, the evolutionary development of this one crucial hidden file is clearly evident. Version 3.1, for example, didn't support the DRIV PARM function for defining 720K (and other) drives that weren't anticipated by the ROM BIOS of most earlier machines. The most significant change in 5.0 is in support for memory use beyond 640K.

DOS 3.1 (9,564 bytes)	DOS 4.01 33,518 bytes	DOS 5.0 33,430 bytes
BREAK	BREAK	BREAK
BUFFERS	BUFFERS	BUFFERS
COUNTRY	COMMENT	COMMENT
DEVICE	COUNTRY	COUNTRY
FCBS	DEVICE	DEVICE
FILES	DRIVPARM	DEVICEHIGH
LASTDRIVE	FCBS	DOS
SHELL	FILES	HIGH
	INSTALL	LOW
	LASTDRIVE	NOUMB
	MULTITRACK	UMB
	REM	DRIVPARM
	SHELL	FCBS
	STACKS	FILES
	SWITCHES	INSTALL
		LASTDRIVE
		MULTITRACK
		REM
		SHELL
		STACKS
		SWITCHES

1-1 Comparison of the significant functions supported by IO.SYS in 5.0 and two earlier releases of DOS.

And these are just the changes visible to the casual snooper. They represent only the tip of the iceberg and the ever-changing role of DOS. In some cases, it's been a two-way street, at that. DRIVPARM, mentioned earlier, appears first in

version 3.2, was subsequently dropped in 3.3, but then restored again in some later 3.3 versions.

Meanwhile, MSDOS.SYS has changed as well, though the changes there aren't as easily defined. Between 3.1 and 5.0, it's grown from 27,760 to 37,394 bytes, but size can be misleading, as in the case of IO.SYS, which actually shrank in size between 4.01 and 5.0 while adding functionality and features. And so DOS marches on.

Bring in the interpreter

All commands entered at the DOS prompt are passed to the command interpreter for processing. At that point, the command interpreter checks first to see if the command matches any of a set of functions it can handle directly—functions like COPY or DEL or DIR. If so, the interpreter passes the command on down to the operating system. But not as something like *COPY*. It's translated into a call to certain interrupts and coded functions of the operating system down at the hidden level.

If, on the other hand, the interpreter finds no match among its internal functions, it then looks to the current default directory, and then to the path (if any) for a matching command name. If it finds a match there, the interpreter loads and executes the specified program—which, in most cases, communicates directly with the kernel or the BIOS to perform its particular job. Even in the package Microsoft sells as DOS, these *external* functions greatly outnumber those that the interpreter can manage internally.

External DOS commands include such vital functions as FORMAT and CHKDSK, and a bunch of utilities like XCOPY, BACKUP, and REPLACE. And then there are your applications and other utilities. The command interpreter makes no distinction between DOS and nonDOS functions. Everything external to the interpreter gets equal treatment by it, whether the command is SK to load Sidekick, or DOS's own FORMAT. Essentially, then, there is nothing sacred to DOS except the hidden files.

What? Not COMMAND.COM even?

You'll note that, other than for just a passing reference, I haven't mentioned COMMAND.COM in discussing the role of the command interpreter. The omission of COMMAND.COM was no accident, but rather to further emphasize the point that DOS—the real true DOS, the quintessential heart of DOS—is all in just those hidden system files: MSDOS.SYS and IO.SYS (or IBMDOS.COM and IBMBIO.COM with PC DOS).

To prove the point, if you try to boot from an otherwise bootable disk on which you've renamed or deleted COMMAND.COM—a floppy, preferably—

DOS will simply tell you:

Bad or missing command interpreter

The DOS kernel will always look for COMMAND.COM by default, but if it doesn't find it, DOS will look for anything that fits the job description. And it really isn't all that fussy—not as long as it finds something it can recognize as an interpreter.

This is a book about DOS 5.0. so I'll limit the discussion here to just what Microsoft supplies with that particular version. I mention the fact that alternatives exist only to help clarify what is and isn't crucial to DOS as an operating system, i.e., what you can and can't change and still be running under DOS 5.0. With that in mind, let's take a closer look at COMMAND.COM and some of the hows and whys.

Internal vs. external

As mentioned earlier, the first thing COMMAND.COM does when you enter a command is to check it against its own internal list of functions for a match. COPY? Yup, it's there—along with all the other functions and commands listed in Fig. 1-2. And this listing includes only internal functions managed by COM MAND.COM. There's also that other list, in Fig. 1-1, of internal functions that are managed directly by IO.SYS. These are, however, more functions than commands, and many of them relate to functions that can be accessed only from the CONFIG.SYS. Still, they are intrinsic to the operating system as we know it.

BREAK	DIR	LOADHIGH	RENAME
CALL	ECHO	MD	RMDIR
CD	ERASE	MKDIR	SET
CHCP	ERRORLEVEL	NOT	SHIFT
CHDIR	EXIST	PATH	SHIFT
CLS	EXIT	PAUSE	TIME
COPY	FOR	PROMPT	TRUENAME
CTTY	GOTO	RD	TYPE
DATE	IF	REM	VER
DEL	LH	REN	VERIFY
			VOL

1-2 Commands currently supported internally by COMMAND.COM. With the exception of LOADHIGH and LH abbreviations, this list is unchanged from 4.0.

In the above list, you'll find most of the more basic file and disk management commands, and batch processing commands. There's even an undocumented TRUENAME command—not new to this release but still undocumented and discussed later in this book.

Most of DOS's most basic file and disk management commands are there, however. Most, but not all. Look again. FORMAT isn't in there and CHKDSK isn't there—two commands that come a whole lot higher on my list that CHCP (Change Code Page), for instance. Why? Essentially, it's quite simple. COMMAND.COM handles all the functions in that whole list of internals, returns any of a number of error messages when necessary, and manages the loading and execution of any executable file you call for—all in all, a total of only 47,867 bytes in this release.

FORMAT.COM alone is 32,945 bytes all by itself these days—and there are over 200,000 bytes of other .COM files and about a megabyte just in .EXE files in this release. There's no way that all of this could be squeezed into memory at once.

So it's a trade-off, and even at that the size of just COMMAND.COM and the DOS kernel has grown by leaps and bounds. To put things in perspective, it takes more RAM just to load DOS today than the 32K or even 64K most of the early PCs had when they left the factory. Ignoring for the moment the fact that, in this release, DOS can load most of its own essential bulk up into the High Memory Area on 286 and higher machines (giving you more room that most users have ever seen to run their applications, much more than with DOS 3.x typically), even the part that must go inside somewhere is still a lot of DOS. And if it won't fit there . . .

On the outside

Once the command interpreter has checked against its own internal list, found no match, and had to go look elsewhere, one file looks pretty much like any other. There is, however, a hierarchy. Unless the file extension is specified in the command, COMMAND.COM checks files with a .COM extension first. If there's still no match, it checks .EXE files, and only as a last resort looks for .BAT files before reporting *Bad command or filename*.

.COM, .EXE, and .BAT—these are the only valid extensions for executable files when running under DOS. DOS 4.x also used a .MOS extension for some executables, particularly in conjunction with the SHELL, but this was dropped in version 5. The default installation of DOS 5.0 includes some 40 .COM and .EXE files, or about 10 percent of all the .COM and .EXE programs on my hard disk.

So how does DOS tell its own external functions from anybody else's programs if COMMAND.COM looks for only certain file extensions and everybody writing programs has to use the same extensions? Very simple; it doesn't. DOS doesn't even try to differentiate because it really doesn't matter. Not to DOS—not to the real DOS, the kernel. The kernel doesn't even seem to care that much about extensions, given the increasing number of times DOS interchanges .EXE and .SYS, etc. Once COMMAND.COM accepts an executable file as valid and loads it into memory, the kernel will try to follow its instructions.

And now we're really getting to the heart of the matter because, at this point, COMMAND.COM pretty much drops out of the picture. It's done its job: it has interpreted the command and found a valid matching file (program) and loaded it. From this point on, everything is pretty much between whatever program you've loaded and the kernel. Even when it comes to performing straight DOS functions.

Straight to the heart of DOS

Say you're working with your word processor and you want to save a file to disk. COMMAND.COM doesn't even have a save-to-disk command. You can play some games with COPY, but that's doing things the hard way, so most word processors—and spreadsheets and databases—go directly to the kernel with instructions, like which disk drive to access, and how to get about the operation.

Just as the entire command interpreter can be replaced by a surrogate interpreter, functions normally internal to COMMAND.COM can as easily be managed from any other program that provides the necessary calls to interrupts, etc., at the system level. It is, in any event, the kernel, not COMMAND.COM or another program, that does the actual work. Programs only tell the kernel what to do and when.

From all of this, you might—in fact, you should—begin to see some semblance of a picture starting to emerge. Basically, if you could communicate with DOS directly, you could throw away however many megabytes of executables and really get things done a whole lot faster. Faster, that is, if you were fluent in machine language. Don't be in any rush to dump all your software, however, because it isn't quite that easy.

The important thing to understand at this point is the nature of the various levels that exist just in the operating system, and the way they fit into the grand design of DOS and computer operation.

CHAPTER 2

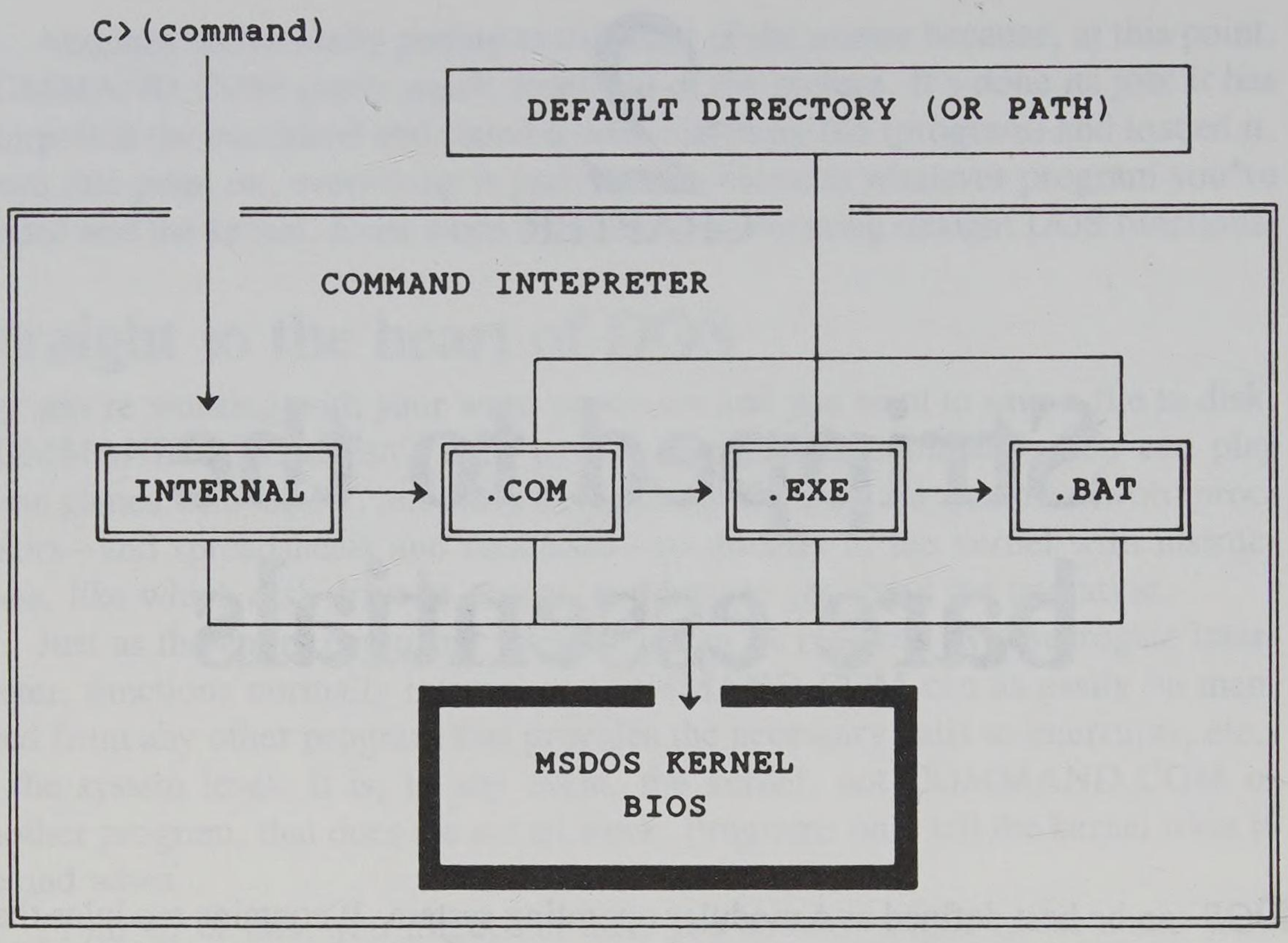
Stripped to the bare essentials

DOS can be best defined as a modular operating system. It contains modules that are themselves composed of lesser modules, some of which are used and then discarded during the boot cycle—which, in part, accounts for the disparity in the size of the files DOS loads from, as well as the amount of space DOS occupies when loaded.

Most books of DOS define three primary modules, but in order to understand any operating system—and to better see its relationship to the overall operating environment—you mustn't lose sight of the crucial roll played by the ROM BIOS that the machine is given by its manufacturer, which is, in fact, the foundation upon which DOS or any other operating system has to build. For your purposes here, therefore, consider the operating system as consisting of the four modules shown in Fig. 2-1.

Here, as you see, you start with a module that's totally independent of DOS—or any user-installed operating system. This is the ROM BIOS I mentioned in the previous chapter—one or more read-only memory chips installed by the manufacturer at the factory, containing the basic input/output services instructions. It's to the ROM BIOS that the awakening computer must look during the boot cycle for instructions on how to even find DOS—or any external operating system—and then load at least enough of that external system to allow it to complete the installation process itself.

Unfortunately, the crucial role of the ROM BIOS is misunderstood by many—if not most—DOS users. But because it's the foundation on which DOS—or any



2-1 DOS's four-module operating system.

alternative operating system—must build, I'll take a moment to put this key player in the scheme of things into perspective.

Basic I/O services

One of DOS's most vital functions—and one it does remarkably well—is moderating conflicts that would otherwise occur between your application software and the dozens of IBM look-alikes or *clones* on the market. Although most people use the word *clone*, there is, in fact, no such thing—and if there were, IBM would make quick work in court of getting any such machine removed from dealer's shelves. So to fairly assess DOS's virtues and shortcomings, it's essential that you understand exactly what DOS has to try to do.

In the beginning, there was really only one machine—the original model of the IBM PC. That machine, used the way it was intended to be used by its creators, didn't need DOS or any external operating system. Today, no real computer can run without some kind of supplemental operating system. Take away the software we call an operating system from even the most sophisticated computer, and all you have is a bunch of chips and transistors that can't add 2 + 2 . . . Add?—what's that?

Every time a computer is turned on, it must be taught even the most simple, basic things—whether it's the first time out of the box or the ten thousandth. And

the creators of the original PC endowed it with everything it needed to get up and running at the factory. Turn one on and it would boot, then put you automatically into BASIC. All the instructions the PC needed to interpret keyboard inputs, everything it needed to know to operate, and even a programming language to let the user play with the computer were built right in.

The way that these initial functions were—and are—built in is via ROM chips, or Read Only Memory. These contain all of the BIOS instructions needed to get the machine started. All that's needed is a little default switching in the computer that, once the computer has contemplated its own navel and whatever else awakening computers contemplate when first awakening, points the way to the ROM BIOS in order to start downloading the instructions needed to turn that pile of chips into something you can use.

Don't believe it? Just take the ROM BIOS chip out of your computer some time and try to boot DOS—or any other operating system. Without the ROM BIOS, your computer can't even *find* DOS—and if it did, it wouldn't know what to do with it.

So, in practical terms, it's the ROM BIOS and not DOS that's the true operating system. There's nothing primitive about the concept of ROM-based operating systems. DOS itself, complete with its external functions, is available in ROM and, in that form, is ideal for many applications. DOS is really nothing but a superset, an add-on. Or, at least, that's how it started. Over the years, DOS has taken on a far bigger role, but to understand what DOS is today, you need to understand its rude beginnings and its original role.

Enter DOS

In any event, it's to the raw hardware that makes up the system and the foundation created by the ROM BIOS that you must add the operating system as you generally think of it—DOS, XENIX, OS/2, or whatever.

With DOS, the rest of the real operating system is contained in just two hidden files—hidden in the sense that one of the file attributes, something I'll talk about a little later, is set in such a way that an ordinary call for a directory won't see it. Other than that, a hidden file is just like any other file, and can be unhidden if need be. These two files, IO.SYS and MSDOS.SYS, in conjunction with the system BIOS, make up the entire operating system.

The *entire operating system*. Period. That's it! Of the something over eighty files that come with the DOS 5.0 package, the whole ball of wax comes contained in two. Take a minute to digest that raw bit of information because it's important to understand this if you're going to understand exactly what DOS is—and what it isn't.

Those two files, working in conjunction with the system BIOS, provide the basic mechanism for every function DOS can perform. Any increase in that func-

tionality must be provided for either in the system BIOS or these two files, as well as the access to the underlying hardware they provide.

At a glance, there seem to be exceptions to that rule. DOS can't come out with a new, upgraded version every time some oddball, unsupported piece of hardware or some radical new concept in computer design comes along. Yet new concepts and hardware pop up all the time, and we keep using them with DOS.

Such things, however, at least at first, are interfaced only by installable device drivers—drivers generally available only from the same 3rd-party source who hatched the idea in the first place. So if you look close, it really isn't DOS that's doing it, but rather something that's hanging on to DOS's coattails for the ride. What I originally said still stands.

Maybe later, if the gods are willing and if the idea really catches on, Microsoft will add support to DOS's repertoire the way it did with 3 1/2-inch floppy drives in version 3.2. And maybe not. What's important here, though, is to note that, even before you get past DOS's most inviolate (almost) inner workings, you're talking 3rd-party add-ons into picking up the slack.

You mustn't lose track of the fact that, no matter what Microsoft might want to add to DOS, DOS can't do it all. The underlying hardware must support it or it won't work. The machine you buy today could conceivably support functions neither DOS nor any other operating system in existence today requires or can even use. That's fine. The point is that DOS can work only up to the physical limits of the hardware and its access to that hardware through the BIOS. And, at the user level, one step removed, you can work only up to the limits provided by DOS. The limits as determined by the two major DOS files—and sometimes a little 3rd-party help.

This isn't to say that DOS—or any other operating system—is necessarily going to fully utilize all the hardware capabilities of each machine. Those capabilities keep changing, as companies keep adding new technologies and new devices. It's the hardware that must set the pace. It's for the software to attempt to keep up.

This is why there must be new versions periodically. You can run DOS 2.0—probably even 1.0—on any of the new machines. DOS 2.x generally can't even find the hard disk on today's machines, but they can be booted from a floppy. You wouldn't want to, however, because you'd be wasting much of the potential built into many of the new machines—things that no one even dreamed of in their wildest dreams when 2.0 came on the scene.

So, at the heart of DOS, you have two files—two hidden files that do the work. What about all the other files that come with DOS? They talk to these two files—one or the other or both, depending on the task. But only within the range of services and functions provided by these two files.

The loader

It's not enough to simply provide a supplemental operating system. Until you actually load it—install it, if you will—you have nothing. That might seem obvious, but the complexity of the process involved is not, and while most of us will never have to deal with a system at this level, a little understanding of some of the basics makes it easier to put DOS and its relationship to your overall system into perspective.

I've established that, as originally conceived, the PC would require no supplemental operating system. It was totally self-contained with all the operating system its creators imagined it might need built right in at the factory on a ROM chip. Not surprisingly, then, when you turn on your computer, the first thing it does is look to the ROM BIOS for instructions. This is sort of like gulping down that first cup of coffee in the morning—you might be moving, but you're really not alive yet, and maybe won't be until about your second cup.

In your computer, however, that "second cup" is made of stronger stuff. But to get to that, the ROM BIOS must tell the system to go look for it. By default, the first place it looks will be drive A:, always drive A:. Watch your system while it's booting, and at some point—after the ROM BIOS has done about as much for you as it can do—you'll see the drive light on drive A: come on and maybe hear the drive mechanism hunting. And only when the system, prompted by the ROM BIOS, can't find anything to boot from in drive A: does it look to drive C:—or, in the case of the old PC that wasn't supposed to really need outside help, back to the ROM for alternate instructions. The ROM built into genuine IBM machines would, at that point, complete the boot cycle by loading a form of BASIC contained on ROM. Providing a BASIC kernel in ROM remains a carefully protected proprietary feature that distinguishes IBM machines from clones, one that requires *all* clones to use a DOS that's somewhat different (and includes a free-standing disk-based BASIC) to achieve functional compatibility.

The ROM BIOS routines, being essentially rather crude, don't examine the contents of a disk—any disk—in detail. The ROM instructions tell the still groggy system to sort of stumble over and check sector zero of track zero. If there are further boot instructions there, execute them, otherwise move on, and if all else fails just sit and sulk—or issue some cryptic error message. Sector 0 of track 0 (computers always count from 0 rather than 1) of any disk has a special significance, then—while physically no different than any other sector on the disk and perhaps containing other data totally useless to the system at this stage, it's the only place boot data can be located in order for a particular disk to continue the boot process. Not surprisingly, this is called the *boot sector*.

The boot sector, like any other sector on a disk, contains only 512 bytes of data, however. Not kilobytes, bytes—less than it takes, for instance, to store just

seven lines of 80-characters-per-line text on disk. Not very much considering that, running under DOS 4.01, for example, your minimum up-and-running operating system requires some 66K of RAM. And the boot sector contains only 512 bytes, which can't spill over into sector 1 because that's reserved for the beginning of the File Allocation Table (FAT). Not very much—but enough, hopefully, to point to a *loader*, and, through it, to the files that are the real heart of DOS, stored elsewhere on the disk. The term *loader* is also used in conjunction with a load module contained in executable files. Although the functions they perform are somewhat similar, these two kinds of loaders are functionally quite different.

The instruction set contained in those 512 bytes is generally referred to as the *bootstrap loader*—the term *bootstrap* from the old saying about pulling yourself up by your own bootstraps. At this point, even though the system has no idea where the rest of DOS is—or what it even looks like if it stumbled over it, the bootstrap loader knows. So it goes out and finds the essential parts of DOS and reads them (loads them) into RAM.

These parts of DOS that are searched out consist of MSDOS.SYS and IO.SYS (equivalent to IBMDOS.COM and IBMBIO.COM in PC DOS)—this far into the boot cycle is the first you'll even see recognizable DOS filenames. And even within this process, there is further bootstrapping because MSDOS.SYS contains a special module called SYSINIT, which initializes certain functions (buffers, system tables, etc.) and then discards itself. So now you're up and running, but what do you have?

Beyond just basic I/O services

Loaded from the hidden file IO.SYS (or IBMBIO.COM), the DOS BIOS forms the primary interface between the hardware and the DOS kernel. This is the supplemental BIOS for the rudimentary I/O services support by the system ROM. It's here, particularly prior to DOS 5.0, that you'll find significant differences between releases of essentially the same DOS versions supplied by different OEMs. Equipment manufacturers have always been allowed (under the terms of licensing agreements with Microsoft) to customize this file, and this is important because the BIOS support needs of machines from different vendors differ significantly. There should be less of the kind of problems this has produced in earlier releases with the new "one size fits all" DOS 5.0 platform.

In areas where the instructions provided by the ROM BIOS or the version of DOS are sufficient, the presence of the DOS BIOS might be completely invisible to the user. IO.SYS automatically loads a default set of device drivers, anticipating the possible existence of certain devices (that might have also been anticipated by the ROM BIOS) whether they, in fact, exist on the system or not.

It's where those ROM-based services don't suffice—or have been superseded by later hardware or other needs not anticipated by the ROM BIOS—that the role

of the DOS BIOS becomes more visible, and more crucial. (This can also be one of the most compelling reasons for upgrading to DOS 5.0, even with older machines.)

You can see this by tracing the history of the now popular 3 1/2-inch floppy drives, which didn't even exist when some of the hardware in use today was manufactured. Were recent versions of DOS unable to overwrite the floppy support provided by the ROM chips in these older machines, it wouldn't be possible to install drives of this newer format without physically replacing the original ROM.

Otherwise, a machine designed in the days when the 360K floppy was king would still deliver only 360K from the smaller drives. In lieu of either ROM or DOS support, installable device drivers can also be used to support a wide range of otherwise unsupported devices. Such device drivers for otherwise unsupported devices are supplied with many upgrade kits. Upgrading to a higher version of DOS, however, might eliminate the need for such installable drivers if the supported devices have become fairly standard in the meantime—as has been the case with 3 1/2-inch drives. In version 5.0, that support now specifically includes 2.88Mb floppy drives, anticipating hardware not even generally available, let alone in use at the time of its initial release.

When all else fails, you can still use installable device drivers to provide an interface for unsupported devices—you can be sure that any vendor marketing an unsupported or novel device is going to be sure to provide a means of interfacing with it in the absence of direct support by DOS. But the pattern keeps repeating. First the hardware changes to take advantage of new technologies and user demands. And then the software must play catch-up. It's a vicious circle.

The DOS kernel

Unlike the supplemental BIOS (IO.SYS), the DOS kernel supplied by Microsoft as MSDOS.SYS has remained the same for all OEM releases of the same numerical version. There might in fact be minute differences, but only to correct problems discovered after the initial release. The kernel occupies a place between the user interface and the BIOS in the pecking order, neither dealing directly with the user or with the hardware. Yet some of DOS's most important business is transacted in the areas of:

- process control (program loading, execution, etc.)
- memory allocation and management
- peripheral support
- the filing system

As you can see, all these functions are, in themselves, independent of the system hardware. At some point, they must be communicated to the hardware, but that's not the kernel's problem. It communicates with the BIOS and it's up to the BIOS to sort out when, how, and why what goes where.

Given that, it isn't hard to see that, at this level, you shouldn't see any difference, say, between Tandy, Compaq, AST, Delta Gold, or any other vendor label. And when we speak of DOS, it's essentially this one file that, for better or worse, makes DOS what it is. The DOS kernel is the final arbiter between the user at the one end of the chain, and the actual hardware at the other—and protected from the vagaries of both, in one case by the command interpreter and in the other by the BIOS.

Isolated and insulated from these distractions, the kernel quietly goes about its business. So let's take just a brief look at just exactly what that business is.

Process control

Process, as defined in this context, includes such things as program loading, execution, and termination; scheduling; and interprocess communication.

DOS is actually more sophisticated in this area than most users probably realize. For while it's essentially not a multitasking operating system, it can have multiple programs loaded into conventional memory at the same time—as long as there's enough memory to support them. DOS doesn't recognize expanded memory, even though access to it—the page frame—is within DOS's one-megabyte address range. It isn't, then, that your TSRs possess any special extra-DOS powers. It's just a matter of programmers availing themselves of powers DOS had all along (or at least going back some time).

Still, considering the fact that DOS is not and never was intended as a multitasking system, it does have a few "hooks" that allow certain external programs to do multitasking on their own, dividing the CPU's attention between two or more applications (time slicing). DOS's own PRINT.COM is such a program, performing limited concurrent processing by taking control of system resources when DOS is idle. Microsoft's Windows, DESQview, and some other operating environments add further task switching support.

None of this is true multitasking, however, because the 8086 family can manage only one task at any given time. But by switching back and forth fast enough, there is an illusion of multitasking while, in fact, two separate and unrelated processes are being carried to completion. If either process is "CPU intensive," as in the case of complex and lengthy computations, there will be a significant slowing of the system when multitasking. However, where neither application is especially demanding, the effect is often not noticeable.

Piping, as supported by DOS—something I'll look at in some detail later—is also managed at this level.

Memory management

Not to be confused with the level of memory management provided by this release for 80286 and higher systems, this is memory management at a very basic level.

The amount of memory required to load different programs varies greatly and might even vary during program execution.

If you had only one program loaded at a time, there would be no problem. Either it would fit or wouldn't. You can't run with just one program at a time, however, because DOS itself is nothing but a close-knit family of programs, and DOS has to be loaded in order to load anything else. So even just to load one program, you have to do some memory management.

DOS provides memory management based on a pool of variable-size memory blocks. DOS has to allocate memory from that pool to a program when it loads. And when the program terminates, DOS returns the allocated memory to the pool. Individual programs might perform some additional memory management for their own needs, but only within limits as prescribed by DOS.

Peripheral support

The operating system provides peripheral support to other software. It does this by making available a uniform set of calls by which that software can communicate with and through DOS, using the system's various resources.

At one time, it was common for such calls to be made by the application directly to the devices, but this limited the ability of those applications to access only a limited number of specific devices. By making any necessary translation a function of the operating system, not only is the role of the application greatly simplified, but the number of devices that can be accessed—and not just by a single program, but by any program—is greatly enhanced.

The level of support required depends on the nature of the device. Keyboards, monitors, and printers require little of the operating system beyond forwarding to the appropriate device. Clock devices require translation of the data into time and date formats compatible with the operating system in use—in this case, DOS. Disk drives and other block devices require probably the highest level of operating system support due to the number and complexity of operational functions involved.

The file system

The file system traditionally makes up one of the largest portions of an operating system. This type of system is built around the use of some form of storage media. And, believe it or not, it was largely DOS's superiority in this area that edged out CPM in early PC days when IBM would sell you either one.

For a disk to be usable, the operating system has to format it. As vital as this function is, I'll discuss it in much greater detail in a later chapter, but for now suffice it to say that the operating system has to mark the media magnetically in certain ways so it can go back and find whatever files it might have stored there.

This marking is done during the FORMAT process, and must include some

basic information about the system, a place for allocation information, a directory structure, and space for the files. This is a pretty rigid operation. The floppy disks you buy are labeled *soft sectored*, but that simply means the media as supplied is perfectly plain and the same disk can be magnetically marked with tracks and sectors to suit quite different file structures imposed, say, by Apple and by DOS machines, or even by a variety of disk drives with different format capacities. And once it has re-created its own file structure on the media, the operating system has to manage access to that file space.

As is the case with other operating systems, DOS can also map (redirect) I/O to character devices as if they were files. It can also redirect I/O from other than the normal default character devices to files, with the COPY CON command.

Through the years, this part of DOS has been greatly enhanced, but the basic, underlying system is essentially unchanged—and can't change without obsoleting every bit of software out there, to say nothing of the chaos it would cause with files. Not to worry, though. It isn't going to happen and, in fact, the same basic scheme has carried over into OS/2.

This, then—these two modules, DOS's hidden files and the ROM BIOS, *are* the operating system. Keep in mind, however, that, no matter what, the hardware has to take the lead. All that DOS or any add-on operating system can hope to do is follow.This is why DOS has—as Microsoft assures us—continued to upgrade to meet the challenge of our ever-changing needs. In DOS 5.0, they've gone a long way to fulfilling that promise, it would seem.

CHAPTER 3

The final puzzle piece

As you can see, an operating system—any operating system—reduced to it lowest common denominator is really nothing more than a translating device. Or, more specifically, a device that translates your wishes into terms a machine can deal with. But even before you reach the level of the actual operating system, you need a middleman of sorts, because even such elementary commands as COPY, DEL, and DIR have no meaning down at the working level. Every command you enter must be interpreted. And so there's yet another level—the command interpreter or, in DOS, COMMAND.COM.

There's more to COMMAND.COM's job description than merely translating your wishes into machine language, however. Translation works only as long as COMMAND.COM itself can understand what you're asking, and it doesn't really know that much. Its limited vocabulary is shown in Fig. 3-1. If it isn't one of these commands, COMMAND.COM can't do it. But it can look elsewhere. If a match is found, COMMAND.COM loads and *executes* that program. And it's in that latter capacity that COMMAND.COM is of the greatest service—more as a traffic director than an actual doer of great deeds.

If you were to enter something on the command line like BURP, COM MAND.COM would first check against its limited command list. Finding no match, it could only presume that what you entered was a command, and COM MAND.COM would go looking.

```
BREAK       DIR         LOADHIGH    RENAME
CALL        ECHO        MD          RMDIR
CD          ERASE       MKDIR       SET
CHCP        ERRORLEVEL  NOT         SHIFT
CHDIR       EXIST       PATH        SHIFT
CLS         EXIT        PAUSE       TIME
COPY        FOR         PROMPT      TRUENAME
CTTY        GOTO        RD          TYPE
DATE        IF          REM         VER
DEL         LH          REN         VERIFY
                                    VOL
```

3-1 Commands supported internally by COMMAND.COM in 5.0 include new support for loading TSRs above 640K on 386 and higher systems.

The pecking order

It's important to understand that COMMAND.COM always does things in a certain order—simply finding a match isn't enough. COMMAND.COM *always* tries first to find a match by checking all files with .COM extensions. If no match is found, it then checks for a match among the .EXE files, and finally, as a last resort, among the .BATs.

This is important for several reasons, not the least of which is that it's possible to have .COM, ,.EXE, and .BAT files all with the same name in the same directory. It won't happen often—but it does happen, as shown in several examples from my own hard disk in Fig. 3-2 (one case is deliberate, but the other poses a potential problem).

```
DV          .COM      9,746 .A.. 1-09-90   2:26:00 am
DV          .EXE    131,924 .A.. 1-09-90   2:26:00 am

VMAP        .COM      9,728 .... 1-15-88   8:00:00 am
VMAP        .EXE      9,760 .... 4-19-88   2:20:00 am
```

3-2 Two examples of both .COM and .EXE files with identical names—one a deliberate programming trick, the other accidental but resulting in DOS running the wrong (older) program.

In the first case (DV.COM and DV.EXE), DESQview is designed to run directly from the .EXE file for some types of systems. For other systems, the working configuration is modified to take advantage of certain other hardware features. DV.COM isn't supplied on the distribution disk as such. Rather, it's distributed as XDV.COM, which, when needed, is renamed DV.COM during installation. This assures that it will always be called before the .EXE file, even though both files coexist on the same directory. The .COM file does whatever it has to do, and then loads the .EXE file. Calling an .EXE file from a .COM file isn't an uncommon programming trick—any more than calling either (or both) of these

files from a batch file—although you'd more often see them given different filenames.

The other case that's shown in Fig. 3-2, where both .COM and .EXE files turned up with identical filenames, presents an interesting paradox. The latter file—the .EXE file—is, in fact, an updated version of VMAP, one of the better disk optimizers in my opinion. But because it has a different extension, it didn't overwrite the older file when installed. So the older file is a .COM file, while the newer file is .EXE. So when will DOS ever load the newer of the two? Never—.COM files first after internals, remember? Fortunately, in this particular example it didn't seem to make much difference either way, but DOS's rigid pecking order can create some interesting problems for the unwary.

Reading between the lines, there's another good argument here for keeping your software compartmentalized into separate directories as much as possible, because the shorter the lists of eligible files on the default directory and directories in the path, the faster COMMAND.COM can complete its search. Which brings us to the next part of the equation—the path.

Even the road to "you know where" is paved with good intentions

Except for its immediate environs—the default directory, whatever that is—COMMAND.COM checks only the places you've specifically marked for it to look. This is where the path comes in.

We all know about the path and what it does . . . And here there's another classic case of what a dangerous thing a little knowledge can be. PATH has to be one of the most abused and ill-used DOS functions there is, and if you can't screw it up bad enough left to your own devices, a lot of software you install will gladly muddy things up even more.

Aside from just cluttering your environment, an issue I'll address in a later chapter, the path can slow you down instead of speed things up. Unless you use a little common sense, that is.

Again, as when checking the default directory, COMMAND.COM plods laboriously through every swamp, dead-end, or mudhole in your specified path in sequence, reading from left to right, repeating the same multi-level search for a precise match. If you have six or eight different directories specified, it will take a much longer time to find something out there in that farthest-out location, certainly, than something close at hand. And even the slightest infraction like misspelling a command will send DOS searching to the farthest ends of its little world before it can even report *Bad command or filename*.

Even assuming the program you're looking for is there somewhere, going through your entire path to find it is often not the best or fastest way to access it—particularly now that you have the new DOSKEY macro capability. Much like a

batch file, but stored in RAM for almost instant execution, macros can even have the same names as the files you want to access. A macro called EDITOR, for example, could literally cut right to the chase by telling DOS to go directly to C:\DOS\XYWRITE\EDITOR, which is the program file for my word processor. (See information about batch files and DOSKEY macros later in this book for more details.)

This isn't to say that you should try to do away with the path altogether, by any means. Used correctly, it's one of the most valuable file handling tools you have at your disposal. But keep it short.

Batch file processing

Here, COMMAND.COM plays a double role, not only bearing the responsibility for finding and executing batch files, but also providing the mechanism for them—even to the extent of supporting a command structure almost exclusively intended for batch file use. In fact, some command, GOTO for instance, can be used only in batch files. For the most part, they can be entered directly from the command line—a trick that can be particularly valuable in perfecting lines that will be included in batch files.

Nobody ever claimed that batch files were a high-powered programming tool. But even with just the following few rudimentary commands, it's really quite surprising sometimes how much you can accomplish with a little ingenuity:

CALL	GOTO
ECHO	IF [NOT]
ERRORLEVEL	PAUSE
EXIST	REM
FOR	SHIFT

For those unfamiliar with batch files, there's nothing mysterious about them. A batch file, reduced to its simplest form, is nothing but one or more DOS commands contained in a text file that has a filename ending with a .BAT extension. Invoking the filename at the DOS prompt will cause the commands contained in the file to be executed sequentially.

You can create simple batch files directly from the keyboard at the DOS prompt by redirecting output to a file, as I'll discuss a little later in the book. Longer or more complex batch files are usually better created, however, by means of the new DOS 5.0 editor, EDIT, or a word processor. The AUTOEXEC.BAT is typical of such files.

While batch file processing capabilities in DOS have remained fairly constant through a number of releases, dating even to the early days, CALL, added in recent versions, has increased the flexibility of batch files greatly. With CALL, it's possible to, as the name implies, call one batch file from another with control

returning to the calling program on completion. This allows you to use smaller batch file modules, particularly where branching and conditionals are involved.

As simple as it might seem at first glance, batch file processing is still too complex a subject to be covered fully in this book. There are, however, several excellent books devoted entirely to the subject. Throughout this book, however, I'll periodically refer to and present simple batch file tricks that are pertinent to specific subjects. And in a later chapter I'll look at at least a few of the more advanced techniques.

The great—or not so great—communicator

DOS's command interpreter is also a communicator—an interface not just between you and DOS, but between DOS and you. If DOS hiccups, it's generally the command interpreter that tells you (with the exception of a number of hardware errors that, when they happen, are reported directly from IO.SYS).

You can see this for yourself with a little harmless poking around inside COMMAND.COM, either with DEBUG, the Norton or Paul Mace Utilities, XTREE, or any of a number of others. Obviously, you won't be able to make heads or tails of most of what you're going to find there, but there's a surprising amount of plain English that should help to give you a better understanding. Figure 3-3 shows a dump of what you'll find browsing through just one area of version 4.01's COMMAND.COM.

The error messages you'll see are all those nasty things DOS loves to throw at you at the slightest provocation, and most of them are contained in COMMAND .COM. Some, in addition, are buried in the hidden files (mostly IO.SYS). But most of them are in COMMAND.COM. And a little farther down is a considerably shorter list of what COMMAND.COM can to do for you—the internal commands I discussed a little earlier.

Just as everything that DOS can possibly do is provided for down at the level of its hidden files, the same is true for COMMAND.COM—and visibly so. COMMAND.COM isn't perfect but, for most users, it's certainly adequate. So now the stage is set. The last puzzle piece is in place. It's time to see what DOS 5.0 really has to offer.

Part 2
Installing and configuring DOS 5

For the first time ever, upgrading any DOS machine—IBM or otherwise—running on 2.11 or any later release is via a special retail upgrade kit distributed through normal software outlets and supported directly by Microsoft.

And that isn't all that's changed. MS-DOS 5 introduces a vastly different new INSTALL procedure that preserves whatever version of DOS you're currently using, including all of your current configuration files, ready to restore should you encounter any difficulty with your new installation. This is about as safe and foolproof as it gets.

But there are *two* DOS 5's out there. In addition to the upgrade kit, there will continue to be an OEM DOS distributed by hardware vendors for use on new machines. These OEM versions, while functionally identical once installed, incorporate a completely different installation procedure and should *not* be used to upgrade existing systems.

Part 2
Installing and Configuring DOS 5

CHAPTER 4

Installing DOS 5

Setting up for DOS 5.0 is a snap. The SELECT process that had become so cumbersome in 4.x is gone, replaced by a quite painless SETUP program in 5.0. You can also install DOS 5.0 from either the A: or B: drives, which can be a big help sometimes. And it's of course available on both 5 1/4-inch and 3 1/2-inch disks—five 360K or three 720K. Which is a lot of DOS.

There are actually two MS-DOS 5.0 releases—three, if you include PC-DOS as a separate OEM release. The one most users will likely encounter is the upgrade version. The same upgrade kit will upgrade *any* DOS machine, including those currently running IBM PC-DOS, and it's available for the first time through many software distributors, as opposed to from the OEM distribution, like earlier releases. Priced at under $100, this release can be installed only on machines already running under some earlier DOS release, as none of the upgrade disks are bootable.

For new installations, there is also an OEM release of 5.0 that differs from the upgrade version only in the installation procedure and possibly the price. There's a separate PC-DOS OEM release available through IBM. PC-DOS 5.0 is functionally similar to MS-DOS 5.0, but, like prior releases, is intended to run only on genuine IBM machines. The most notable difference will be found in the new BASIC interpreter, which, in the PC-DOS release, will (as with BASIC and BASICA in the past) again run only on genuine IBM machines, as opposed to the free-standing QBASIC supplied with MS-DOS 5.0. Also, some utilities might have different names. For new installations, the first of several setup disks is bootable in the OEM release. There are, however, a number of other significant differences in the installation procedures that need to be understood and appreciated, especially because the OEM version lacks most of the amenities found in the upgrade kit release.

Upgrading from an earlier DOS version

The installation program supplied with the upgrade kit, SETUP.EXE, is remarkably easy to use—you might even go so far as to say foolproof. Maybe not perfect, but close, very close.

This should be especially appreciated by anyone who went through the old SELECT routine, which got increasingly complex and confusing in prior releases. By the time DOS 4.0 through 4.01 came on the scene, it was really the pits. Mercifully, it's gone in this release.

One of the most important features of the upgrade kit version is that it won't even install until and unless you let it preserve your existing DOS installation by creating a self-booting uninstall floppy using the system files from your old DOS version.

This is more than just an ordinary bootable floppy, however. In addition to your previous basic operating system, it has all the data needed to reconstruct the FAT, root directory, and other crucial disk data in an absolute worst-case scenario. And to undo whatever has been done, this UNINSTALL floppy, as you can see from the directory listing in Fig. 4-1, even contains UNINSTAL.EXE, a special program invoked by the one line that AUTOEXEC.BAT DOS writes to this floppy.

```
 Volume in drive A is UNINSTALL 1
 Directory of  a:\*.*

autoexec.bat        10   12-13-90    4:09
autoexec.dat       871   11-27-90   15:02
bios.old         33129   10-26-90    5:38
bpb0.dat           254   12-20-90   12:49
command.com      46246   12-13-90    4:09
command.dat      45466   10-26-90    5:38
config.dat         403   12-04-90   19:14
dos.old          37378   10-26-90    5:38
fat.dat          32768   12-20-90   12:49
global.dat         703   12-20-90   12:49
mboot0.dat         512   12-20-90   12:49
pboot.dat          512   12-20-90   12:49
root.dat         16384   12-20-90   12:49
setup.ini        10397   12-13-90    4:09
uninstal.exe     88560   12-13-90    4:09
wina20.386        9349   10-26-90    5:38
      326,144 bytes in 16 file(s)
      813,056 bytes free
```

4-1 Directory of the self-booting disk created by the upgrade kit installation program. If used, AUTOEXEC.BAT automatically initiates the uninstall process to restore pre-5.0 DOS configuration.

Everything else—all the files in your DOS directory—are moved to an OLD_DOS.*n* from which they can be recovered and restored to their original directory should you want, for any reason, to return to using your prior version and configuration. Having prepared for the worst, rather than writing over all of your old DOS files with 5.0 files, SETUP moves them to OLD_DOS.*n* so UNINSTAL.EXE can easily restore them too, if need be.

If SETUP should find something on your disk like OLD_DOS.1, left over from an earlier installation, rather than overwriting it, SETUP will create an OLD_DOS.2, etc.,—yes, subdirectories with extensions—but don't you try it.

This is a mandatory step and 5.0's upgrade SETUP won't proceed beyond this point until and unless you give it a disk to save your old system on.

Note, however, that this uninstall feature is found *only* in the upgrade kit version. Because the OEM version is intended primarily for new installations that would have no such pre-existing configuration, it will write over any system that might exist, simply wiping out the prior version along with any configuration files. This is the major difference between the two distinctive versions of this release.

Lovers and other strangers

There are some scenarios that could make recovery of your old DOS an important option. For instance, as with any new DOS release, there will be some applications software that isn't happy. Mercifully, the problems with DOS 5.0 are minimal from all indications, due in part to SETVER, a neat new DOS device driver SETUP which automatically installs in the CONFIG.SYS, and can fool many programs into thinking that they're actually running under one of the older versions of DOS—whatever older version they happen to like. So don't let the .EXE extension fool you. SETVER is a device driver and, while certain of its functions are available from the command line even when not installed as a device driver (in much the same way as they would be if it was an ordinary .EXE file), it performs its intended purpose only when properly installed from the CONFIG.SYS. And a number of software developers resolved conflicts with this new version while 5.0 was still in beta, prior to public release.

Frankly, though, even the first beta version Microsoft let out on this one—almost a year before public release—was surprisingly trouble-free and what few problems it still had were mostly put to rest some months before its actual release. I hesitate to say *all* because surely there must have been some minor bugs that did slip though, just waiting for the infinite number of busy fingers at an infinite number of keyboards.

By providing a "back door," however, DOS 5.0 allows for the quick return of your prior system, complete with all startup files exactly as they were. This allows you to continue work uninterrupted, while contacting the developers of such soft-

ware regarding availability of an upgrade that does support 5.0. So for anyone dependent upon using software that for some reason might prove incompatible, this option could prove invaluable. Another big plus for this release.

Go for it

Secure, then, in the knowledge that there's a way out, you can install DOS 5.0 as an upgrade by simply loading the SETUP program from the first of the several distribution disks and, at the DOS prompt, typing SETUP and hitting the Enter key. Unless you abort the installation for some reason, it manages the whole operation from that point on. It does pause, requesting that you verify its analysis of your system and checking to see if you want the DOSSHELL loaded automatically when you boot your system (in which case it adds it to your AUTOEXEC.BAT). It also gives you an opportunity to back up your hard disk before proceeding, which is always a good idea if you don't have a current backup anyway.

Other than that, it just quietly goes about its business, prompting only when it's time to change disks. Regardless of which drive you chose to install from, drive A: *must* be available and you must have at least one free 720K or greater capacity floppy disk (or two 360K) when prompted to insert a disk in drive A:. You should, of course, make sure that you not only have a disk that's bootable with your old system, but also a drive A: that the disk fits in, because many of us have A: and B: drives of different sizes these days.

You can reuse old disks if you want. However, any prior files or data on the disk(s) you use will be lost because the SETUP program reformats on the fly so it can make the floppy self-booting with your old operating system if you should suddenly decide you want your old DOS back.

The whole upgrade installation typically takes about 10 minutes, but that will vary depending on machine specifics. On an old 4.77 Mhz 8088 with an appropriately slow old hard disk, it took just under 15 minutes. It also depends on how closely you attend the process.

Let setup do it for you!

Any installation of DOS 5.0 from the distribution disks *must* be done through the SETUP program—and this is true of both the upgrade kit and OEM versions. This differs from previous releases such as 4.01 that could be installed manually by using the Escape key to quit SELECT, the automatic installation program that comes up whenever you boot from the first distribution disk.

There's a reason for this. With this new release, Microsoft is, for the first time, supplying DOS in compressed form. The only files not in compressed form are SETUP.EXE (which isn't copied to your hard disk) and the few other files—mostly those necessary to manage your initial installation.

This isn't the popular PK compressed format being used by so many software

developers these days—the one that packs a number of compressed files into one big big lump that fills the screen with messages about "exploding" this and "expanding" that. Here, the individual files are compressed and the last character of the extension is changed to an underline character, _. So you have a lot of .EX_, .CO_ and .SY_ files, and the fact the files are decompressed during installation is quite invisible. But, for example, 9509 bytes of UNDELETE.EX_ unpacks to create the 15,925 UNDELETE.EXE.

So the first time around, you have to install your new DOS the way Microsoft wants it done. After that, you can treat it just like any of the earlier releases, using either the SYS command or formatting with the /s option to create bootable disks. And, of course, once decompressed and given back their proper extensions, the DOS utilities can simply be copied from disk to disk like any other files.

This isn't to say, if you decide to prune the something over 2Mb that the installation disks unpack into your DOS directory down to just the DOS utilities you need, that you need to worry about erasing any you're not completely sure about. There's a new DOS EXPAND.EXE utility that will expand any of the compressed files—even the hidden files—individually, at any future time they might be needed. Expanded in this manner, the R, S, and H attribute bytes aren't set, nor are they installed to the disk area necessary to make a disk bootable.

While the documentation says you need 2.8Mb of free disk space to start the upgrade installation, a full, non-networking hard disk installation takes some 2,094,279 bytes out of your disk space, which, though well short of the 2.8Mb, is still a big chunk. If you just plain don't have a couple of megabytes of hard disk space to spare, SETUP does support a /m switch that does an absolutely bare bones installation—the two hidden files and COMMAND.COM only. Here, too, you could use EXPAND.EXE later to expand just the utilities you need. That could get pretty tedious, however, and, in most cases, you'd be better off to make the 2.8Mb that DOS wants to see available on your hard disk, even if it means transferring a bunch of things to floppies temporarily to clear the space.

Speaking of floppies, the upgrade version also supports an /f switch, enabling you to do the whole DOS 5.0 installation to a series of floppy disks if you like—seven 5¼-inch 360K or four 3½-inch 730K floppies. Although not documented at this writing, because the first disk is the startup disk, the process can be terminated at the point you're prompted to insert a second fresh disk to install to. The first disk will, at that point, contain a bootable system, although it might not contain many of the utilities you want. This could also to get around the need for so much free hard disk space.

The CONFIG.SYS file

The SETUP program will add at least one new device driver, SETVER.EXE, to your existing CONFIG.SYS (or create one if you don't already have one), as well

as a SHELL= statement. SETVER is an interesting new feature in 5.0 that can fool some programs with minor incompatibilities into thinking they're running under some earlier version. Both changes are benign, but you you might want to check them and, under some circumstances, delete the SHELL= statement. For users running 4DOS (or some other alternative command interpreter) rather then COMMAND.COM, this is especially important—4DOS, by the way, works fine with 5.0.

Through the last version of DOS available for testing before this went to press, there was one peculiarity that, though not serious, should be noted. As with all peculiarities, I repeated the installation procedure to verify that it wasn't a fluke. When installed on a 386 with no CONFIG.SYS, SETUP created the following file:

```
DEVICE=C:\DOS\SETVER.EXE
FILES=10
SHELL=C:\DOS\COMMAND.COM C:\DOS\ /p
```

However, when installed with a CONFIG.SYS that installed a 3rd-party memory manager already in place, SETUP made the following changes:

```
DEVICE=C:\DOS\SETVER.EXE
DEVICE=C:\DOS\QEMM\QEMM386.SYS ....
.
.
.
.
.
SHELL=C:\DOS\COMMAND.COM C:\DOS\ /p
DEVICE=C:\DOS\HIMEM.SYS
DOS=HIGH
```

The purpose, of course, is to relocate some 47K, the bulk of the 5.0 kernel, to the High Memory Area (1024K-1088K). It's surprising that HIMEM.SYS didn't try to install itself when no prior CONFIG.SYS was present, especially because you can have only one memory manager installed and, when you boot, DOS loads the first one that it finds. You can have only one manager for each of the specific memory areas. However, because most of the better memory managers today are actually total memory managers that incorporate inseparable support for extended and expanded memory, the High Memory Area, and the ability to map memory to upper memory blocks, the use of one generally precludes the use of any other.

Note, though, that these lines show up at the *end* of the CONFIG.SYS file if you already have one. This is important and just one more little kindness Microsoft has built into DOS 5.0. For 80286 and 80386 (and higher) owners currently

using some other extended memory manager—QRAM or Move'em, for example, on a 286 or 386MAX or QEMM on a 386 or higher—this means that the SETUP program doesn't try to push any form of memory manager you might currently be using aside. Rather, when you reboot, if by the time DOS reads the HIMEM.SYS it has already loaded someone else's manager, it simply skips on past, as shown here:

```
HIMEM: DOS XMS Driver, Version 2.77 – 02/27/91
XMS Specification Version 2.0
Copyright 1988-1991 Microsoft Corp
    Error: An Extended Memory Manager is already installed
        XMS Driver not installed
```

The use of a memory manager other than HIMEM.SYS doesn't preclude the use of the DOS=HIGH option for loading the bulk of the DOS kernel to the High Memory Area, however. This is a function of DOS 5.0 itself (support for loading both device drivers into memory mapped to previously unused address space above 640K is also provided by IO.SYS), and not a function of HIMEM.SYS. This was, in fact, done with several other memory managers at various times during the writing of this book.

In any event, you'll probably want to come back later and at least take a look at whatever changes or additions SETUP has made to your configuration. I'll look at these issues in greater detail when I discuss customizing options and memory management issues later in this book.

Cleaning up

You should, then, be up and running under 5.0 as soon as you're satisfied that you:

- Have a satisfactory installation of DOS 5.0
- Decide that you—and your software—can live with it, and maybe even really like it

You can run a program call DELOLDOS to get rid of all of the old DOS files, free the disk space, and recycle the floppy that DOS saved your old startup configuration to—DOS 5.0 won't let you do even that, however, without pausing and warning you of possible consequences, as shown in Fig. 4-2.

DELOLDOS even removes itself from your new DOS directory in the process. The only caution I might add is that you should also be sure that there are no files other than old DOS files in your old DOS directory that might be moved as well. When SETUP moves your old DOS installation, it moves everything that's in the directory, and when DELOLDOS cleans house, it really cleans, and even the directory SETUP saved them to is gone.

```
Warning

Running Deloldos removes all old DOS files from your system,
making it impossible to recover your previous DOS.

To continue with Cleanup, press Y.
To exit, press any other key.
```

4-2 Warning box displayed by the DELOLDOS command before it deletes the copy of pre-5.0 DOS from your hard disk.

Installing 5.0 on a new machine

There's also, as stated earlier, an OEM version of 5.0 that isn't intended for installation by the general public—or at least not the same general public as the upgrade version. Disk #1 in this set—the same number of disks as in the upgrade release—is bootable, and its AUTOEXEC.BAT puts you right into the SETUP program. Same program name but a totally different installation routine.

The SETUP program for the OEM release has none of the endearing safety features of the upgrade version. If you run it on a machine that already has DOS on it, it simply wipes out the pre-existing DOS. It's gone! Also gone are your AUTOEXEC.BAT and CONFIG.SYS files—although not totally. They're still there, renamed AUTOEXEC.OLD and CONFIG.OLD. You don't want to be in too much of a rush, however, to just write over them because the new CONFIG.SYS created by SETUP adds a new device driver, SETVER.EXE, so you'll probably want to lift that line and add it to your old CONFIG.SYS before dumping the new one and renaming the old one back to CONFIG.SYS again.

Of course, for a new installation, these points aren't even valid. If there's any message in this, then, it's that, if you're upgrading rather than starting from scratch, even if you have a choice, go the upgrade route. It's easier and safer.

One last difference between this and prior OEM releases, which should be noted by experienced DOS users. As in the past, the installation program can be aborted from the first screen that comes up after the system boots. Unlike earlier releases, however, you can't at that point transfer the DOS 5.0 system to a disk because SYS.COM, at that stage, is SYS.CO_, a compressed file that isn't executable until it's expanded during the installation process. There's a way around this, but it isn't worth the effort.

On a machine that doesn't have a hard disk—many laptops, for instance—the OEM release SETUP routine will, sensing that there's no hard disk to install to, automatically shift into a floppy-install mode, but this isn't an option if a hard disk is present. Here, too, the floppy install process can be interrupted after the first new disk in the new installation is created because, at that point, you have a bootable DOS 5.0 system disk, although it might not contain all the utilities you'll need. You can now, however, expand the program from the distribution disk using

EXPAND.EXE, found in ready-to-use form on one of the distribution disks.

You will not be tempted to use the #1 installation disk as an operating disk because it's write protected for anything but installation use (on the 3 1/2-inch disk, the write protect slide had been completely removed to discourage any attempt to alter or eliminate the AUTOEXEC.BAT that forces it to load and run SETUP). You can, of course, terminate SETUP from the first screen by hitting the F3 key, but you don't have much that's usable to work with at that point—only FORMAT, UNFORMAT, and SETUP. That's about all—unless you count NLSFUNC.

For use on networks

DOS 5.0 adds a level of network support not seen in previous versions, as well. Whether working from the OEM or upgrade release, you'll be prompted to give a Y/N-type response to whether you'll be running on a network. Answering Y will cause the setup program to expand and install several network redirectors that won't be installed for non-network installations.

There is a rather minimal level of support, however, for use with LanMan or Novell. Extensive DOS 5.0-specific support is also available from Microsoft for other network applications, for an additional fee.

Reaching for new highs

Regardless which of the installation packages you have, once the SETUP program gets past giving you one last chance to back up your files and gets down to serious installation, it pretty much just goes ahead and does things the way it pleases. Neither the upgrade or OEM versions automatically installs FASTOPEN or any of 5.0's auxiliary TSRs the way 4.x's SELECT did, but both always leave you with a CONFIG.SYS file of some sort. If SETUP hasn't done it for you and you have an 80286 or higher machine, you'll probably want to edit your CONFIG.SYS to add the following lines (the new EDIT feature of DOS 5.0 is ideal for jobs like this):

```
DEVICE = HIMEM.SYS
DOS = HIGH
```

For most purposes—and most users, probably—the DOS=HIGH option by itself is—or should be—worth the price of the upgrade. With that line in your CONFIG.SYS on a 286 or higher DOS machine, 5.0 will load some 47K of itself to the high memory area (HMA)—this, as discussed in greater detail elsewhere in the book, is that mysterious extra 64K that DOS can access beyond one megabyte. With that and only about 17K of DOS loaded down in conventional memory, you've probably got more memory for even your biggest applications than you ever thought possible—over 40K more than what 4.x ever gave you, certainly.

You have so much memory that programs have never had access to, that some software has encountered problems loading, returning a *Packed file corrupt* error

message—don't ask where that one came from, but at least one of its meanings seems to be that something is having a problem trying to load way down low. Again, however, DOS 5.0 has a solution ready and waiting in another new utility called LOADFIX, which I'll talk about in greater detail later.

Decisions, decisions

This extra user memory and some other memory management tricks the new DOS has up its sleeve are certainly high on the list of 5.0's most important new features. Particularly if you've spent all your DOS life living below 640K until now. However, if you are among the many 386 (and higher) owners already using one of the several 3rd-party memory managers to map memory from whatever pool of extended or expanded memory you have, to unused DOS-accessible addresses above 640K, it might be time for some serious decision making.

At first glance, the choice might seem obvious: if DOS wants the HMA to load itself and can free up that much memory, let DOS have it. For most users, this is certainly the best bet. For those whose application software might already be using the HMA, however, it might not be. DESQview users, for example, might already be using 63K of the maximum 64K available up there, wasting only about 1K versus the 17K (64K−47K) DOS will waste if it grabs the HMA.

For now, just be aware that SETUP will add HIMEM.SYS to your CONFIG.SYS if your machine will support it, but that, if you're already using a memory manager, HIMEM.SYS won't load. This doesn't foreclose the option of loading the bulk of DOS into the high memory area when you boot, however.

Upper memory blocks: memory DOS has never given you before

In addition to being able to move most of its own bulk out of your way, with 386 and higher machines, DOS 5.0 also provides the means of relocating device drivers and TSRs that would ordinarily take up valuable space below 640K into previously unused address space above 640K. Above 640K, but below the 1024K–1088K high memory area, where DOS loads that big piece of itself.

Power users have been using this space for some time, mapping expanded memory to unused DOS address space with the help of 3rd-party memory managers such as DESQview's QEMM or 386MAX from Qualitas, but now, for the first time, DOS itself provides the mechanism with EMS386.EXE. This is a new EMS emulator, which, used in conjunction with HIMEM.SYS, also provides full LIM 4.0 EMS expanded memory support for any software using expanded memory.

EMS386.EXE must be installed manually, however, as must DEVICE HIGH= or LOADHIGH commands to load device drivers and TSRs into upper memory blocks. There are also a number of other customizing features available that must be installed manually, as well, but I'll deal with them later.

CHAPTER 5

Doing DOS my way . . . or yours

DOS, as I established early in this book, tries to be all things to all computers in the 8086 family. Not an easy task—or always even possible—given the tremendous range of hardware variables among so called "compatibles." Or the proliferation of "DOS" software created by thousands of dedicated programmers, most of whom, it seems, not only don't talk to each other—or Microsoft—but often don't even subscribe to the same set of rules. The real wonder is that any of this works at all.

But it does surprisingly well considering what it has to put up with. DOS somehow manages to smooth over most of the many hardware differences, making them invisible not only to you but to your software—and making differences in software invisible to both you and that impossible collection of hardware you call a computer.

For whatever reason, however, there are limits, and at some point you or your software has to intervene. A minor adjustment here to perk things up, some oddball new device out there to be interfaced. And then there's software that inflicts cruel and unusual punishment on the system—DOS needs a little help sometimes to deal with things like that, as well. And that help generally comes in the form of a special file called the CONFIG.SYS.

The CONFIG.SYS

Whether you realize it or not, every time you turn on your computer you must configure it. It's usually a painless and usually invisible process—but it happens. Computers have short memories; deprived of power a fraction of a second at most, they go completely stupid—and even if they haven't, an immediate reboot wipes out anything that might be left in memory.

As discussed earlier, few of a computer's most vital functions install automatically during the startup procedure. Those included in instructions carried on the various ROM chips usually do. In addition to the ROM BIOS, hard disk and other special drive controllers, many video display cards, networks, etc., carry ROM instructions specific to those devices. Others must be loaded—loaded into memory like any program. And even where the ROM contains instructions that are read into memory, they might need to be written over by the operating system to reflect enhancements not anticipated by the ROM instructions. The more complex your system, the more of these software configuration add-ins you've got to deal with at some point.

The CONFIG.SYS is the first—and lowest—level at which your special needs or wishes are considered. That usually makes the CONFIG.SYS one of the most important files in your system. It's the lowest level at which you can control or influence the operating environment or system. As its name reflects, its purpose is to configure your system to meet your needs.

At the most rudimentary system level—say a PC with only one or two floppy drives and a hard drive, but no extended or expanded memory or devices not anticipated by DOS—no special configuration and hence no CONFIG.SYS is even needed. This is an important fact to keep in mind in case you have trouble and need to boot up from a floppy while you try to sort things out. However, by the time you're far enough into computers to even ask the kind of questions this book addresses, you're pretty sure to have one.

Unlike functions loaded from the AUTOEXEC.BAT file—the next thing DOS will look for once it's finished here—programs loaded and parameters passed by the CONFIG.SYS can't be added to or altered once DOS has finished reading and executing the CONFIG.SYS and turned control of the system and whatever operating environment you have created over to you.

A device driver, whether one of DOS's several default ones or one that's user installable, by itself provides only an interface between some specific device and your computer. The interface does not activate the actual device. That's generally done by other software—software installed at a higher user level. The driver provides only the means for that other software to communicate with the device. No device driver, no device.

You might not realize it, but even floppy disks and hard disks require specific interfacing. The reason you might not be aware of it is that DOS simply provides

these services internally. You don't have to install them in your CONFIG.SYS. DOS just assumes you're going to need them—DOS means Disk Operating System, remember?—and installs them automatically. It didn't always, however, at least not to the extent it does today. DOS prior to version 3.1, for example, can't even find my hard disk—yours either, probably. Even optional, installable DOS support for floppies with densities greater than 360K didn't come along until the version 3.x, and support for 2.88Mb 3½-inch disks didn't arrive until this new release.

That doesn't necessarily mean you're going to use all the default device support provided. But DOS installs the interface so that if and when your application needs to read from or write to floppy A: or B: or to an ordinary hard disk, the interface, the enabling instruction set, is in place and ready.

At the CONFIG.SYS level, the various function attributes that can be defined, i.e., the default values overwritten or added to by the CONFIG.SYS, include:

BREAK
BUFFERS
COUNTRY
DEVICE
DEVICEHIGH (386 and higher systems only)
DOS (286 and higher systems only)
DRIVEPARM
FCBS
FILES
INSTALL
LASTDRIVE
SHELL
STACKS
SWITCHES
VERIFY

With that in mind, I'll show you each of these configuration variables individually, along with some of its tricks and traps. The important thing to keep in mind, however, is that if the operating environment, as determined by the items in this list, isn't set the way you want it by the CONFIG.SYS, in most instances your only recourse is to revise your CONFIG.SYS and reboot. In an extreme case—if the system crashes before turning control over to you—you might need to reboot from a bootable floppy to get your system up and running, even enough to be able to edit the offending CONFIG.SYS.

Don't get the idea, though, that, just because you can set values for all these variables, you should—that if you don't have all of them in your configuration,

you're missing something. While DOS needs to know all these things at startup, it isn't an all-or-nothing situation. DOS *assumes* a default value or set for each of these. You need to list only the items in your system that differ from those defaults. And, in fact, you're better leaving any others alone—you know, sleeping dogs and all that . . .

Still, you shouldn't be afraid to edit or experiment with your CONFIG.SYS. Often, in fact, a little tinkering is the only way to get things up and running right. This is especially true when using DOS 5.0 on 286 and higher systems where memory management becomes an issue.

Also, a lot of software installation programs want to change your CONFIG.SYS (and your AUTOEXEC.BAT) to suit themselves. Often times some changes are required, but unfortunately, not all installation programs work the way they're supposed to—at least not on everyone's PC—and when they don't, results are often unpredictable to say the least. This is when you need to understand the nature of the beast and how to handle it.

The CONFIG.SYS is nothing but an ASCII text file and, as such, is easily written, rewritten, or modified by using the new DOS EDIT function. You can even create it directly from the DOS prompt using COPY CON to redirect output from the console to a file, although, except as a temporary expedient to get things up and running, that usually isn't the best way.

BREAK

(The default is BREAK=OFF.) This one has to do with when DOS gives you the opportunity to interrupt or terminate a process (other than rebooting) by pressing either the Ctrl−Break or Ctrl−C key combinations. When successful, BREAK returns you to the DOS prompt to do whatever you might want to do from there—restart the process just terminated or some completely different task.

In the default configuration, every time DOS writes to the screen or printer or reads input from the keyboard, it checks to see if the user has issued one of these commands. If so, DOS issues an interrupt, which normally terminates execution of whatever program is running.

Programmers can and often do restrict DOS's opportunities to check for BREAK commands. However, setting BREAK=ON in the CONFIG.SYS will increase the number of times DOS checks, even including at the completion of every disk I/O operation. Unfortunately, about the best thing you can say for that is that it significantly increases system overhead, making applications run slower. That's the good news.

The biggest danger lies in the fact that the stop is sudden and unceremonious. Breaking out of an application is different than breaking out of a batch file—there, at least, the system usually pauses and questions whether you want to *Terminate batch job? (Y/N)*. Breaking out of a running application can probably be best described as something like jumping from a plane without a parachute. It can lead

to system crashes with some software. Even worse, BREAK=ON can actually dump some programs, leaving files open, databases corrupted, etc., if an unanticipated breakout is encountered.

Such occurrences aren't normally the fault of DOS, but rather something in the software you're breaking out of—a programmer allowing a breakout in the middle of some critical process. In any case, while BREAK=ON can be a handy tool during program development and debugging, for most user applications it's better left OFF.

BUFFERS

(The default varies.) Whenever DOS reads from or writes to a disk, it uses portions of RAM designated as *buffers* to store the data. Each buffer can hold the contents of one 512-byte disk sector—a sector being the smallest block of data DOS disk operations will transfer. Even if your application stores and accesses data or records in smaller chunks, there's no such thing as a partial sector read. Everything that's in that sector gets sucked up into memory at one swipe, and a buffer is where it stores this information temporarily until whatever part or parts of it your application needs are picked up. Writes are essentially the same process in reverse.

Buffers, then, are vital to the efficient operation of your computer. Too few, and your computer often has to wait until it has a buffer free before it reads or writes more data—and when it waits, *you* wait. Yet, if you set up too many buffers, you'll actually slow your system down. That, aside from wasting memory that could be put to better use (half a kilobyte for every unnecessary buffer).

Further confusing the issue, the number of BUFFERS that DOS gives you by default varies depending on your hardware, and particularly on how much memory you have installed. On most of today's machines, you can pretty much assume a default value of 15, but the exact numbers are shown in Table 5-1.

Table 5-1 The recommended number of buffers for different system configurations.

Configuration	Buffers	Bytes
<128K RAM, 360K disk	2	
<128K RAM, >360K disk	3	
128K-255K of RAM	5	2672
256K-511K of RAM	10	5328
512K-640K of RAM	15	7982

There's a fairly easy way to find out if you have too few, too many, or just enough buffers. Make up a simple batch file that runs CHKDSK three times—the actual number of times isn't crucial, but three generally gives you a reliable number to work with. Then sit down with stopwatch in hand—actually, any sweep sec-

ond hand will do. Run the program and note the time. Then change the number of buffers with a BUFFERS=*nn* statement in your CONFIG.SYS—cut back to something like maybe only half a dozen. Then reboot and check the time again. Keep trying different values until you're satisfied. You'll probably be surprised at how clearly performance changes when you reach the point you're looking for.

Paring down the number of BUFFERS becomes a factor especially if you're using SMARTDRV.SYS or any other disk cache software. Caching does much of the work buffers would do otherwise, so if the test above shows you need eight or more buffers, with caching you should have no more than five. This is more than the number the little test above might indicate, but cutting back to only two or three as might seem sufficient—and might well be for some purposes—could very well degrade performance during floppy disk operations.

And before you ask, a larger cache—unlike too many buffers—does *not* degrade performance (except by using up a lot of memory). Buffers are small caches, but they're also "brain dead" caches, where real caches—the better ones anyway—are quite sophisticated, often able to anticipate what you'll need next and have it ready. In any event, BUFFERS= should be set to the smallest number that exhibits no significant slowing of performance.

On the subject of disk caching and the relationship between buffers and caches, note that, beginning with DOS 4.0 and continued in 5.0, BUFFERS supports the addition of a second parameter to specify a number of look-ahead sectors in order to speed disk reads. Previously, this service had been available only with comprehensive disk caching programs. While this enhancement doesn't make adding additional BUFFERS an entirely satisfactory alternative to caching, it does point out a continuing effort on the part of Microsoft to enhance DOS.

COUNTRY

(The default is COUNTRY=1.) Little needs be said about this one because most readers, simply by virtue of the fact that they're reading this in English, will want to use the United States symbol set. That is the DOS default, and for that no entry in the CONFIG.SYS is necessary.

This is just one part of a whole group of multilingual support functions DOS has taken on over the years, as noted elsewhere in this book. Differences have mainly to do with time, date, decimal, and monetary conventions, rather than actual character sets (with the exception of currency symbols). UK users would therefore want to use the United Kingdom set, which requires a COUNTRY=44 entry in the CONFIG.SYS. There are more than a dozen other national symbol sets available, French, Spanish, German, etc., each of which requires a COUN TRY= entry. They are well documented.

DEVICE

(The default is explained below.) Every piece of hardware you connect to your computer is a device: your keyboard, disk drives, monitor—everything. Even things like RAM disks and your ANSI.SYS file, which aren't even physical devices, are devices as far as DOS is concerned. Of course, without at least some bare minimum of things like conventional disk drives, an ordinary keyboard, and a basic display device, you wouldn't have much need of DOS—or much of any other operating system, for that matter. Recognizing that, DOS provides its own limited set of default device interfaces or, as they're more commonly referred to, device drivers.

Beyond that very limited repertoire—often referred to as *standard device drivers*, you're on your own, however. And it isn't even a case of merely expediting communication or smoothing out an innate incompatibility or two. As far as DOS is concerned, without appropriate software interfaces installed, no devices beyond those included as standard in its own limited domain even exist.

A prime example of this is the 384K of extended memory you get when you buy a 286 or higher computer that comes with one whole megabyte of RAM installed. Without adding HIMEM.SYS (or some other special software device driver that recognizes any memory above 640K as a device), it just plain doesn't exists in your PC. Not as extended memory. Not as expanded memory. Not as anything, really. Just as something you bought and paid for.

HIMEM.SYS can access unused RAM as extended memory, however. And to use this portion of RAM as *expanded* memory, you need—that's right—another device driver: EMM386.EXE (usable only with 386 and higher machines). Don't, by the way, let the .EXE extension throw you; it's just DOS playing games with you again—in fact, there are a couple of device drivers in 5.0 with .EXE extension rather than the .SYS you're likely accustomed to from earlier releases. The driver, then, is not something to be taken casually, but is, in fact, every bit as important as the physical (or virtual) device with which it interfaces.

Having said that, it follows that everything pertaining to a device driver is of critical importance in attaining maximum performance. Most device drivers—like DOS itself—make certain default assumptions that might be relevant to your particular system. Typically, a device driver for mapping memory to empty address space above 640K will *assume* you'll be using expanded memory, as well, and not map to the 64K page frame area. If this isn't what you want, you have to add some sort of parameter that tells the driver "no page frame"—that or waste 64K.

As both the devices and their interfaces have become more complex, their software installation procedures, by and large, have become more sophisticated. The trend has been toward automated installation, usually via a separate installa-

tion program—some prompt for user input that allows for certain choices, others are fully automatic. All of which is fine either way, provided things go smoothly.

Unfortunately, however, that isn't always the case, and at that point a little knowledge can make the difference between a relatively mild or severe case of panic. I'll discuss this in more detail in a later chapter, when I deal with software installation practices and pitfalls. You can be almost certain, though, that sooner or later you'll have to come to terms with one or more device drivers in your CONFIG.SYS.

You should also be aware that some devices, by their nature, observe a precise "pecking order"—like building a skyscraper, you can't build the top floor until the foundation and all the intermediate levels are in place. In other cases, the order is unimportant and at your discretion. But it's important to know and understand the difference.

DOS 5.0 offers a perfect case in point. Here, for example, is an actual CONFIG.SYS, including some drivers that must observe a precise pecking order mixed in with some that can go anywhere:

```
DEVICE = C: \ DOS \ HIMEM.SYS
DEVICE = C: \ DOS \ EMM386.EXE 256 RAM I = B000 – B7FF
     i = F800 – FDFF X = C800 – CBFF
DOS = HIGH,UMB
DEVICEHIGH = C: \ DOS \ RAMDRIVE.SYS 720 /E
DEVICEHIGH = C: \ DOS \ SMARTDRV.SYS 360
DEVICEHIGH = C: \ DOS \ MOUSE.SYS
DEVICEHIGH = C: \ DOS \ EGA.SYS
DEVICEHIGH = C: \ DOS \ STAK \ STACKER.COM   /B = C800   /M = 0
     D: \  STACVOL.000
lastdrive = z
files = 20
buffers = 5
```

The first four items must play follow-the-leader on this basis, on two conditions:

- Unless there is extended memory available to work with, i.e., unless HIMEM.SYS has already been installed, EMM386.EXE has no raw pool of memory it can draw from to emulate expanded memory.
- If EMM386.EXE isn't able to emulate LIM 4.0 expanded memory, then there's no expanded memory for RAMDRIVE.SYS or SMARTDRV.SYS. And while the other drivers shown don't directly use either extended or expanded memory, unless HIMEM.SYS and EMM386.EXE are both doing their thing, there's no memory available to map to upper memory blocks—so DEVICEHIGH isn't going work and these will be loaded in conventional memory instead.

All sort of like a house of cards.

DEVICEHIGH

Mentioned previously and new to DOS 5.0, like the more familiar DEVICE command, DEVICEHIGH is a mechanism for loading device drivers from the CONFIG.SYS, differing from DEVICE only in that it attempts to load the specified device drivers into memory mapped to upper memory blocks—if available—on 80386 and higher machines. In the absence of mapped memory—or in case there's insufficient mapped memory available—specified drivers are loaded into conventional memory exactly as they would be with an ordinary DEVICE statement.

The syntax for using DEVICEHIGH is exactly the same as for the more familiar device and examples can be seen in the CONFIG.SYS file. It's a rather benign command, and I would encourage even users unfamiliar with working above 640K to try using this with all device drivers instead of DEVICE—anyone who has a 386 or higher machine, that is, with HIMEM.SYS and EMM386.EXE loaded, and observing the defined pecking order. Exceptions would, of course, be HIMEM.SYS itself as well as EMM386.EXE, both of which must be loaded previous to making any request for UMB usage.

This is a command that someone with a little experience working with mapped memory can really put to good use, realizing increases of up to 100K or more of available conventional memory. Less experienced users probably won't achieve such spectacular results, but even 30K−50K of overhead removed to upper memory means 30K−50K of additional conventional memory available to run applications. That would be well worth the effort in any case, but doubly so because, in this case, you have nothing to lose but the little time it takes to experiment with the command.

DOS

(The default loads the DOS kernel in conventional memory, with no memory mapped to upper memory blocks on supporting machines.) All new in 5.0, this command is the key to DOS 5's new memory management scheme, which includes not only loading most of the kernel to the HMA (286s and up), but also mapping memory to unused DOS address space above 640K (386s and up only), if available.

It isn't functions of the device drivers that do the actual memory management. Rather, like other CONFIG.SYS functions, they're handled internally by IO.SYS. It's important to realize this because, being independent of any particular proprietary memory management software, these services are available in the absence of HIMEM.SYS and EMM386.EXE, and can be used in conjunction with 3rd-party memory managers like QEMM or 386MAX. The DOS command statement can be used in any of three different ways:

- DOS=HIGH, to load the bulk of the kernel to the HMA (above 1024K) on 286 or higher machines.

- DOS=UMB, to map LIM 4.0 expanded memory to unused DOS address space above 640K but below 1024K on 386 or higher machines.
- DOS=HIGH,UMB, when both management functions are required. This can also be accomplished with separate DOS=HIGH and DOS=UMB statements.

Of the two primary functions, the usage of upper memory blocks requires the greater degree of user understanding for optimum performance. It's an open-and-shut case as far as DOS=UMB is concerned—either you have it or you don't. Period. What isn't cut and dried is how much you have to work with and, when dealing with HIMEM.SYS and EMM386.EXE, that amount can generally be enhanced significantly with a little effort, as discussed in the chapter on memory management.

DRIVPARM

(The default is established by machine BIOS.) DOS 3.2 and later, although not all releases, allow you to set parameters for various block devices, such as nonstandard floppy drives (*nonstandard* being defined, in this case, as anything other than what was standard at the time the machine was manufactured and therefore not anticipated by it's BIOS).

Prior to the introduction of DRIVPARM, this could only be accomplished through the use of proprietary device drivers tailored to the specific needs of particular drives. With the introduction of DRIVPARM, however, that changed, allowing the user to set alternate parameters for each drive that might require them, instead of through the use of such special separate device drivers.

While not required with most machines of recent manufacture, this one still serves an important role when working with older machines that might have been upgraded with the installation of newer, higher capacity drives. It would also be required if the new 2.88Mb drives were to be installed in even fairly recent machines. The controller card would also have to support the higher density drives.

FCBS

(The default is FCBS=4,0.) This one is a real dinosaur. File Control Blocks (FCBS) were used by early—repeat, *early*—versions of DOS to track each open file. Unless you're running pre-DOS 2.0 programs, you can pretty much forget this one.

FILES

(The default is FILES=8.) File handles were introduced beginning with DOS 2.0 for file I/O operations. This function replaced FCBS when it was found that many programmers were circumventing that scheme and directly manipulating the

fields of the File Control Block. This resulted in nonportable programs that didn't work under new DOS versions. The changeover to file handles forced programmers to observe good programming practices. At least in that regard.

DOS predefines five of the default eight handles at startup for use by default devices (keyboard, 2 screens, printer, and aux), leaving applications with the ability to open only three files. Unlike the relatively high overhead of BUFFERS, additional FILES require only 48 bytes each over the default number of 8. So it's generally recommended that you increase this number to about 20 (the maximum is 255) in your CONFIG.SYS by adding the line FILES=20.

The number 20 is also the maximum number of file handles any program running under DOS can use. Again, adopting a "more is better" attitude can be costly, although the number of bytes required by each additional file handle provided for is significantly less, at 48, than that required by additional BUFFERS and there's nothing to be gained by exceeding the recommended 20 (unless you receive an error message indicating *No free file handles* or some other message when you try to run specific software).

INSTALL

(This has no default parameters.) A DOS 4.0 addition and recommended by Microsoft at that time, not much is said about INSTALL in DOS 5.0 documentation. Even with version 4.0, any benefits from using INSTALL were nebulous at best, and, while still supported, it now works contrary to DOS 5's preferred memory management model by forcing supported programs to load low (in conventional memory) rather than into memory mapped to UMBs (386 and higher only).

Also, using INSTALL might foreclose other options. Several hard disk optimizers won't optimize disks after certain TSRs (specifically FASTOPEN) are loaded because of possible interaction that can corrupt files and data. Given this, the INSTALL option doesn't look desirable with this release.

Fortunately, the small group of DOS externals supported by INSTALL includes only the five shown below, all of which are executable files that can be loaded from the DOS prompt (or from batch files) at any time. Supported programs, for your information, include:

FASTOPEN.EXE
GRAPHICS.COM
KEYB.COM
NLSFUNC.EXE
SHARE.EXE

Despite the drawbacks cited above, the creators of some sophisticated security-oriented software—Certus (Certus LAN, developed by FoundationWare, 13110 Shaker Square, Cleveland, Ohio 44120, (216)752-8181), for example—make

good use of INSTALL to load software that prevents not only unauthorized file access, but unauthorized use of disk drives, etc. While their key program, like the DOS utilities listed previously, can be loaded from the AUTOEXEC.BAT—and must be when used in conjunction with earlier DOS versions—using the INSTALL option prevents the possibility of someone using Ctrl−C to break out of the boot sequence before their software is loaded and the machine secured from unauthorized access. It should be pointed out, however, that INSTALL is not supported by Microsoft at this time for other than their own applications.

If you want to use INSTALL or experiment with it, you must, as when installing device drivers, include the file extension on the command line, as demonstrated by the following example:

```
INSTALL = C: \ DOS \ FASTOPEN.EXE C: = 50
```

LASTDRIVE

(The default is E.) Changing the LASTDRIVE option can be useful in any of several situations. First, obviously, is if you have more than five physical drives installed on your system. Setting some higher value is also important if you use the DOS SUBST command, because DOS won't recognize any SUBST command for a drive designator above E: unless some higher value has been specified. The LASTDRIVE specified can be any letter of the alphabet through Z, making SUBST a very useful tool for cutting across whole tangled nests of subdirectories to access files.

LASTDRIVE is peculiar, however, and isn't always necessary, even when you're using RAM disks or other pseudo-disk devices such as Squish or DoubleDisk that require more drive designators than the DOS default of five. DOS, for some reason, will in many cases access F: and G:, etc., when requested by 3rd-party devices, while refusing calls from its own utilities, issuing the error message:

```
Invalid drive "d:"
```

Unlike DEVICE= commands, which often must be installed in a particular sequence, a LASTDRIVE= line, in most cases, doesn't necessarily need to precede a CONFIG.SYS entry that installs a device to a drive letter above the default value.

SHELL

(The default is SHELL=COMMAND.COM.) Unless you don't like COMMAND.COM and choose to replace it with some other command interpreter—4DOS, for example—you need never give a second thought to this one. DOS defaults to its own command interpreter automatically unless you tell it something else here, so no CONFIG.SYS line is needed.

There is an exception. If for any reason you still can't live within 5.0's expanded 256-byte environment and you find you must increase its size, this can be done only by adding a /E value to a SHELL statement:

SHELL = COMMAND.COM /E:*nnn*

STACKS

(The default is STACKS = 0,0 for PC, XT, IBM portables, etc., and STACKS = 9,128 for PC ATs.) This one came into being with DOS 3.3 and doesn't apply to earlier releases. Every time a hardware interrupt occurs, DOS must save the current register values in its stack pool before servicing the interrupt so it can restore the system to its previous condition when finished. The stack pool is then made available for the next calls.

However, if too many interrupt calls are made too quickly, DOS will run out of available stack space and crash. When found, necessary additional stack space can be added—crashes resulting from stack failure are usually identified by the error message left on the screen when the system hangs. Doing so, however, uses more memory and might slightly degrade overall system performance. Therefore, the default value of 0,0 (except with ATs) should be left unchanged—no CONFIG.SYS line required—unless you experience an abnormal amount of system crashes. And even then, change the default only if you're using DOS 3.3 or higher.

Even when working with higher versions of DOS and 286/386 systems, no STACKS might be required. And, in fact, some 3rd-party software publishers recommend changing the stacks value to zero by adding the line STACKS = 0,0 to your CONFIG.SYS. Use this one cautiously, but, when dealing with today's more sophisticated hardware and higher versions of DOS, remember STACKS if you seem to have a problem.

SWITCHES

SWITCHES is another DOS option that was poorly documented and even more poorly understood when it was introduced in version 4.0, but has taken on a somewhat greater although still relatively obscure importance in 5.0.

In its original guise, SWITCHES allows the use of software that can't process the scan codes generated by newer keyboards—those having the F11 and F12 function keys and separate cursor keypad. Every key on a keyboard must generate its own unique code—two codes, in fact (there are separate make and break codes). However, some such software will crash if the user inadvertently presses a key whose scan code it doesn't recognize. Using SWITCHES with the /K parameter instructs DOS to ignore scan codes generated by keys not found on older keyboards. The proper form for this function, when required, is SWITCHES = /K.

The /w switch, new in DOS 5.0, is probably of interest to a far greater num-

ber of users. When Windows is installed, on many machines the setup program installs a read-only file named WINA20.386 in the root directory, which isn't where it really should be. This allows you to move it—say to your WIN directory. It's a two-step process, however. You must:

1. Add a SWITCHES = /w command to your CONFIG.SYS file.
2. Add a DEVICE = [*d:*][*path*]]WINA20.386 command to the enhanced section of your Windows SYSTEM.INI file.

VERIFY

(The default is VERIFY = OFF.) This is another function originally available only from the DOS prompt but, beginning with version 4.0, optionally installable from the CONFIG.SYS. VERIFY, regardless of where invoked, is one of the most commonly misunderstood of all DOS commands. VERIFY tests to see that every disk write operation is performed correctly. It does nothing, however, to verify the accuracy or integrity of the data written. The net effect of using VERIFY is a degradation of system performance due to the longer write times required, but no real assurance of file integrity.

The AUTOEXEC.BAT

After DOS has run the CONFIG.SYS and loaded COMMAND.COM, it still has one last thing to do before it turns the system over to you. It checks for an AUTOEXEC.BAT file.

Although optional, it's hard to imagine trying to run without an AUTOEXEC.BAT today—having to manually type your preliminary commands every time you boot your system. Still, it's an option and, in DOS 5.0, it differs little from previous versions, except that there are a few new features you might well want to take advantage of.

Most notable of these is the new DIRCMD variable, used to alter the default DIR sort to take advantage of the greatly enhanced DIR command in this release. Through the use of the SET command, you can change the default way DOS displays directories to any of a couple of dozen different display formats, all of them choices DOS has never even offered before, let alone made so easy to use. The following line from my AUTOEXEC.BAT, for example, first displays an *alphabetized* list of subdirectories, if any, at that level, followed by an alphabetized list of files, pausing to display long listings one screenful at a time:

```
SET DIRCMD = /O/P
```

An environment issue, this will be dealt with in the next chapter, where I'll deal with that little-understood place we call the *environment* in greater detail. This feature, by the way, is available when running 5.0 on any compatible machine and, like virtually all commands that can be run in the AUTOEXEC.BAT, it can

be run independently from the command line—or run again later from the command line to modify an earlier setting.

Of interest only to users with 386 (and higher) systems is the brand new LOADHIGH command for relocating TSRs in memory mapped to previously unused address space above 640K (but below 1024K). In the following example, LOADHIGH is used to relocate MODE into upper memory.

```
LOADHIGH DOS\MODE COM1:9600,N,8,1,P
```

While the term *TSR* usually brings programs like SideKick to mind (and LOADHIGH isn't fussy about which TSRs it loads, as long as there's room), MODE, whether you realize it or not, is also a TSR. It's a fairly small TSR, admittedly, but *any* TSR moved from conventional memory to upper memory means that much more space for your applications and their data files to run in.

While this command, too, can be run from the command line at any time, it's best done up front and ideally from the AUTOEXEC.BAT. As discussed in greater detail in the chapter on memory management, getting the most effective use of any memory that might be available in this area—an area new to many DOS users—is often as dependent upon the order in which programs are loaded to it as the amount of memory available for use above 640K.

And DOS has never made it easier to experiment with new configurations than with this new release. The new EDIT program, discussed in detail elsewhere in this book, is fast and so intuitive to use you might not believe it's really a part of DOS. From that standpoint, DOS really has made it a whole lot easier. But then configuring your system right—really fine-tuning it to get the most, the very best, out of your whole system has probably never been as vital as it is now, either, given all of the new options at your command.

CHAPTER 6

The environment

The word *environment* crept quietly into the lexicon of DOS beginning with version 2.0. It's an area that many books on DOS skip over rather lightly. Originally, the default size for the environment is only 160 bytes—the equivalent of only two full lines of text on your screen—but in 5.0 has been increased to 256. (The default size is 512 bytes in 4DOS.) That should be more than ample—I rarely use even half of it.

But what is it, this *environment* thing? It doesn't show on a directory. It doesn't even have a filename because it's not a file—or even any part of one. And, even if you go snooping into DOS's innermost plain-language secrets, you'll hardly find a mention of it. Yet the environment, whatever it is, is one of the most powerful user tools in DOS's bag of tricks.

All it is is simply a small place set aside in memory. No more, no less. It's probably best described simply as an area where DOS stores certain information—information DOS has to have available, even information that DOS has no current use for.

The environment can be used for many things—including some you can be sure DOS's creators probably never even thought of. They run the gamut, from very important all the way to total trivia. Listed categorically rather than in order of importance, you have:

COMSPEC= (Use of the equal sign when changing environment variables is optional.) This tells DOS where COMMAND.COM (or an alternate command interpreter) is whenever it has to find it in a hurry. It's a little like PATH, but is exclusively for the use of the kernel—sort of the DOS equivalent to having a key to the executive washroom. Unless reset, COMSPEC defaults to the root directory of the disk DOS was booted from.

PATH= The PATH is a fast lane to any executables. The default is blank—it

doesn't even specify the disk DOS was booted from. It's limited in length only by how any characters DOS allows on a single line, but should be kept as short as possible.

SET *variable*= This is another of those chameleon-like things that means different things to different people. To programmers, it can provide a special private path—still another PATH-like mechanism—for accessing data files required by certain programs. In 5.0, DOS itself supports a couple of SET options. To most users, on the other hand, the SET variable allows virtually any string to be established as a value DOS can test against for a variety of creative uses.

PROMPT= This can change the dull default DOS prompt to almost anything you like, including an optional message, the current default directory, time and date, etc. It can also be displayed in your choice of colors (if applicable) and screen locations.

DIRCMD= This is a brand-new variable in this release. Exclusive to the greatly enhanced DIR command, it allows you to establish customizing parameters (typically from the AUTOEXEC.BAT file) in order to change the default directory listing form.

These are the principal things you'll find in the environment. While the default size is small, you can make it almost any size you want, up to 32K. You don't really want to, however. Just like the environment you live in, you can use your computer's environment wisely or pollute it with all sorts of garbage. Fortunately, it's a generally forgiving place and, once established (during the boot process) you can change anything about it. Except its size—that can be done only with a SHELL statement in the CONFIG.SYS. For COMMAND.COM, the syntax is:

SHELL=*d:path*\COMMAND.COM /P /E:*size*

If you exceed the space allotted, you'll get an *Out of environment space* message. However, if you can't live quite comfortably within the default 256 bytes, you're probably doing something wrong. So back up and take a closer look at just what has to be there—and what doesn't.

COMSPEC

COMSPEC is the only thing DOS puts in the environment automatically. And this one is an absolute *must* because it tells DOS where it can find the command interpreter—COMMAND.COM or any acceptable substitute you might care to use. If you don't tell it otherwise, DOS will assume that COMMAND.COM is in the root directory of whatever drive you booted from, and point a finger that way any time DOS needs to find it.

For instance, if you boot from a floppy disk in drive A:—even one that has no CONFIG.SYS or AUTOEXEC.BAT—and immediately give a SET command (SET and Enter), DOS will return something like:

```
PATH=
COMSPEC=A:\COMMAND.COM
```

DOS did it: created the beginnings of an environment all by itself. All it enables DOS to do is find its way back to COMMAND.COM, even if you're someplace far removed from COMMAND.COM at the time.

There can be a perfectly good and valid copy of COMMAND.COM in the root directory of C:—or even extra copies in every subdirectory. It doesn't matter. COMSPEC says it's in A:\ and that's where it goes looking. That's the *only* place DOS will look for it and, if you've removed the floppy, or for any reason COMMAND.COM isn't where COMSPEC says it should be, the system shuts down, telling you:

```
Not ready error reading drive A
Insert disk with \COMMAND.COM in drive A
and strike any key when ready.
```

And until you do as DOS insists, it will simply sit there, unable to continue. If for any reason you can't do that, your only option is to reboot the system. For while the COMSPEC, like everything else in the environment, can be changed in mid-session, it can be changed only at the DOS prompt. And if DOS can't find COMMAND.COM or a reasonable substitute, you don't even have a prompt. You don't have anything.

Despite the fact that DOS sets a default COMSPEC to the root directory of whatever drive you booted from, your command interpreter can be anywhere you'd like to put it. All you have to do is overwrite the default with a COMSPEC= statement of your own, usually in your AUTOEXEC.BAT. To keep your root directory pared down to the minimum, you could, for instance, keep COMMAND.COM in a directory with all the rest of your DOS files (except the hidden DOS system files, which must be in the root) by specifying:

```
COMSPEC=C:\DOS\COMMAND.COM
```

This doesn't even have to be the same location indicated in the SHELL= statement (if you have one) in your CONFIG.SYS. Or, for that matter, does it even have to point to the same command interpreter. DOS won't let you invoke COMMAND.COM from a different version than the one currently loaded. But you can, as mentioned earlier, boot with COMMAND.COM and switch mid-session to 4DOS or some other alternative command interpreter (or vice versa). Not that

you would probably want to, but understanding that should help you to understand how loosely DOS's inner workings are tied to any particular user interface.

PATH

Since version 2.0, DOS has allowed users to define a search path for executable files not located in the root (or default) directory. And they can then be found from wherever you happen to be, i.e., whatever disk and directory happens to be your current default.

While initially limited to include paths predefined by the user, beginning with version 3.0 the power of the PATH statement was enhanced greatly to allow DOS to find any executable file if you simply type the path ahead of the filename at the command line (or within a batch command), as in the example below. (There are some exceptions, particularly in the case of a small handful of programs that need to access special startup files, but which don't support the use of SET to provide a pathway):

```
LOTUS\123
```

Executable files only—.COM, .EXE, or .BAT. Nothing else. No data files, even if they belong to executable files included in the PATH statement. And this works only when the executable file—Lotus 1-2-3 in the example—is either free-standing or at least able to find any auxiliary files it might need (overlays, etc.). Some can't—XyWrite, a word processor popular with many professional writers, can't be satisfactorily loaded except when the directory where it resides is the default directory. Still, PATH is a very useful and powerful file management tool—one of DOS's best.

In the normal sequence of events when a command is given, typically either at the DOS prompt or from within a batch file, the command processor checks first to see if the command is one of its internal commands. If not, DOS goes looking for something external. But not far. Not beyond the root directory unless you've given it specific instructions to—and then only where you tell it to look.

This makes PATH one of DOS's more powerful file management tools, when used correctly. But using it correctly requires a little knowledge of how it works—because, particularly when making a PATH statement from a batch file, you can create a real monster that your system might choke on. DOS-stranglers like this:

```
PATH=C:\DOS;C:\DOS\UTILITY;C:\DOS\LOTUS;
C:\DOS\BATCH;
C:\DOS\XYWRITE;C:\DOS\JUNKYARD;
C:\DOS...
```

Broken down, every part of that overstuffed creation might be relevant, possibly even essential at *some* point in a normal day's operations (though probably not). But to find something in the JUNKYARD subdirectory, DOS has to wade labori-

ously through no less than, in this case, five subdirectories before it finds the place you told it to look. PATH is slavish and checks sequentially.

There are two lessons to be learned from this about establishing your path from a batch file. The first is to put the directories most often accessed first in your PATH command, with those of decreasing importance at the end.

And the second is that, if they're not important enough to list up front, do you really need them cluttering up your path at all? In many cases, you're far better off calling programs—even ones that could be "pathed" to—via batch files that keep you from having to repetitiously type the path ahead of the command every time.

If you then invoke each of the programs you want to run concurrently from individual batch files, those batch files can start by redefining the path, as shown in Fig. 6-1. This is especially valid when working in a multitasking environment where each task you load also loads a secondary copy of the command processor and inherits whatever environment you established for the primary copy, through PATH, SET, and PROMPT. In those cases, your best bet is to put no more clutter in the master environment than is needed just to get the bare-bones system up and running.

6-1 Batch files that start new applications, like this one, can redefine the path to meet only the needs of that application, in order to speed execution.

```
PATH=/newpath
new_application
```

In a multitasking situation, this redefinition of the path affects only the secondary command processor that's loaded when you call up an application. It has no effect on other windows. Thus, the command processor for each program can run with maximum efficiency in a stripped-down environment.

Unfortunately, the more efficiently you divide your disk into separate directories for different kinds of software, the more a problem the length of your PATH statement can be. And, just as unfortunately, few users seem to really know how little needs to be included in their PATH statement and when certain needs are better served in other ways.

This trick takes advantage of the simple fact that the same filename can exist on two (or more) directories simultaneously, either with the same or different extensions. You can, for example, take the name of an .EXE program and create a .BAT file with an identical command name. If you simply have the .BAT file specify the path for the .EXE program and issue the program name, when you type the name of the batch file, in effect you're echoing the command but adding the complete path to the .EXE file. All that needs be included in the path is the drive and directory where you keep your batch files. And, if by any chance the program you want to execute happens to be in the default directory, that's no problem. DOS will simply find the .COM or .EXE file you want and execute the program without even bothering to look for your clever batch file.

Still, PATH can—and should—play an important role in your file management strategy. But in the interest of efficiency, its use should be limited. Typically, the PATH set by your AUTOEXEC.BAT file should include whatever directory your DOS files are located in (if not the root directory), and the one you keep batch files and possibly utilities in. Much more than that is generally not necessary.

PROMPT

One of the things DOS has traditionally been criticized for is its cold, impersonal nature—some call it downright unfriendly. There's something intimidating about a screen that's completely blank except for a blinking cursor, impassive and impatient beside a couple of cryptic characters—C>.

That's it—just C>, firmly anchored in an inscrutable sea of black. Hit Enter without entering anything and that C> thing will be repeated, down the left side of the screen as often as you hit the Enter key.

Little wonder that cunning shells and cutesy menu systems have multiplied until you're almost overwhelmed with cleverness—often at no small cost, either, to your pocketbooks or precious pool of memory below 640K. There's surely a place for clever shells and menu systems, but not for everything. And surely not to the extent that many users rely on them. For all the nicer a complex and "sophisticated" menu system might appear, the more a reasonably experienced hand at DOS can generally run rings around it just from the blinking cursor. And although it might strain the imagination a bit at first, DOS really doesn't mean to be patently unfriendly. Just frugal—very frugal.

Fortunately, that stupefying C> (or A>) prompt is one of the easiest things to fix in DOS, and an excellent—and completely safe—place even for beginners to assert themselves and make DOS work for them. For instance, if you're into using subdirectories at all, one of the first things you'll probably want to change about the prompt is to have it show you just where you are at the moment—your current default directory, as well as just the default drive.

To handle that—and a number of other customizing features—the prompt command supports a number of special character codes, shown in Table 6-1, that are unique to it.

To get DOS to tell you where you are, you only have to use the $p code—p for path—and type PROMPT $P at the C> prompt. DOS will instantly return C:\DOS: (or whatever directory you happen to be in) as your new DOS prompt. It will remain your prompt, then, until you either issue a new PROMPT command or reboot your system.

The PROMPT command is not case sensitive, so you can use either upper- or lowercase. In most cases, you'll probably want to use two or more codes together to create a prompt that both shows you where you are and also does something

Table 6-1 Special codes recognized by PROMPT command when creating custom prompts. Note that all are preceded by dollar signs.

Code	Display
$q	= (equal sign)
$$	$ (dollar sign)
$t	Current system time--displays time at prompt, but doesn't update
$d	Current system date
$p	Current drive and path
$v	MS-DOS version
$n	Current drive
$g	> (greater-than sign)
$l	< (less-than sign)
$b	| (bar or pipe character)
$_	linefeed
$e	ASCII 1Bh (the Escape character)
$h	Backspace to erase a character on the prompt line (displays the cursor only)

else. PROMPT allows that. In fact, you can do pretty much what you like with it, even spreading your prompt over several lines, if you like, with the help of the linefeed code.

Aside from the special codes you can use to customize the prompt, you can humanize it by adding plain English (or foreign language) messages, like *The current drive and directory are:*. To do that, simply type that message on the command line after the PROMPT command, and follow it with the appropriate code ($p):

PROMPT The current drive and directory are: $p and Enter

and DOS will return your new prompt:

The current drive and directory are: *d*:\[*path*]

You can change the text and the code, change anything you like as long as you start with the word *PROMPT*. If you want to get really fancy, you can even do it in color by mixing in some ANSI.SYS codes, provided ANSI.SYS was loaded from the CONFIG.SYS. (Other suitable ANSI drivers supporting the same code set are also supplied by various 3rd-party sources, some loadable from the command line rather than from the CONFIG.SYS.) ANSI codes are generally prefaced by Escape characters, for those you use the $e to represent the Escape character. Going that route, you might do something like this (no Return, except at the very end of the command):

```
PROMPT $e[1;36m Your current drive and directory are $P$_$e[1;33m
It is now $t. What is your next task?$_$e[1;37m
```

On a color monitor, this prompt would look this—this with the first line in cyan, the next in yellow, and input in white:

Your current drive and directory are *d*:*path*
It is now *time*. What is your next task?

How's that for civilizing DOS? And of course, once you get a prompt worked out that suits you, you can set it up automatically by including an appropriate PROMPT command in your AUTOEXEC.BAT.

SET

As indicated earlier, there are really several ways SET is used: by some programmers as a means of pointing their programs to their data files, or by users for inserting strings having special meanings to them into the environment as a means of giving DOS something to test against—I've even seen it used this way as a rather primitive means of adding password protection to files, although there are certainly better ways.

In this new release, DOS itself provides new services through the use of the SET command to help you customize your system, most notably the DIRCMD variable.

DIRCMD

Brand new in 5.0, this is the one that makes the greatly enhanced DIR command finally do directories the way you want them—well, if not exactly, at least close). While the default directory listing format is still the same old unsorted mess with files displayed in the order in which they were created (until DOS has to start recycling directory entry space) there should be enough sort options now to satisfy the pickiest of users.

Making DOS sort directory listings to your personal liking requires entering one or more special codes that are unique to DIR in DOS 5.0. These can be entered from the command line, and the syntax is simple enough:

DIR [*path*] [/*switch*][/*switch*][/*switch*]...

Of course, typing all that every time you want to call up a directory listing would be a real nuisance. That's where DIRCMD comes in—you just use SET to load the directory sort parameters you want as your personal default into the environment this way:

SET DIRCMD = /*switch*[/*switch*][/*switch*]...

The documentation now lists as many as 27 switches that can be used and combined in various ways, and the total number of possibilities is staggering. Don't let

it throw you, however. There will probably only be a couple that are of real interest to you—most likely the the same two I use:

/o	Displays any directories at that level sorted alphabetically at the top, followed by an alphabetical listing of your files.
/p	Pauses after each screenful is displayed until you're ready to read on.

If you like that, stick this line in your AUTOEXEC.BAT file:

```
SET DIRCMD=/o/p
```

and you've got it made. As for those odd times when you want things sorted differently (although in nearly a year I haven't found one yet), you can issue a new SET DIRCMD= command from the command line with a new set of parameters for DOS to use for the remainder of that session, or, on a one-shot basis, simply tack them onto the end of a DIR command.

DOS 5.0 also supports the use of the SET command to specify special paths for certain temporary files, but these are generally of lesser interest and are documented where applicable.

Down the garden path

Until now, the way most users have come in contact with SET has probably been in situations where SET works sort of like PATH—except for data files. It differs in several ways, however, how different depends on which programs use this function of the environment. MEMO (Broderbund's MemoryMate), for instance, absolutely requires a SET command—either from the AUTOEXEC, another batch file, or manually when it's loaded. Without the following line in the environment:

```
SET MEMO_DIR=d:\path\
```

MEMO won't be able to find its database and index files, not even if *all* of MEMO's files are in the default directory. This is one of the more extreme cases, but such situations do exist.

More commonly, you'll find programs that use SET statements but can do without. Procomm and some other communications packages use SET statements, often inserting them in your AUTOEXEC.BAT during installation, whether you want them there or not. So do many other programs, but in most cases you need them only if you need to specify the path to the program. If the files are in the default directory, no SET statement is needed in the environment and you can remove any related SET command that the installation program might have tacked onto your AUTOEXEC.BAT.

Keep in mind, however, that SET statements don't just happen. They're not created without good reason—or at least *somebody's* good reason. So you can't just start deleting them unless you understand why they were put there in the first place, and make other arrangements. Batch files similar to the one below:

```
CD C:\DOS\TELECOM\PROCOMM
PROCOMM
```

can be used to load most programs that otherwise require a SET statement. A simple file like this assures that the programs will be able to find its files, and when you're finished and move on to something else, you don't leave litter in your environment.

When you must use SET statements, it's important to note, however, that other ENVIRONMENT variables—PATH, COMSPEC, PROMPT, and DIRCMD—overwrite and replace any and all previous values, or strings. SET strings, on the other hand, simply attach themselves to your existing environment, as shown here, where I start with one SET *string* = and add another (checking the contents of the environment with SET commands before and after). At C:\DOS, type SET and hit the Enter key. Then specify:

```
COMSPEC=C:\COMMAND.COM
MEMO_DIR=D:\MEMO\
PATH=C:\;C:\DOS;C:\DOS\UTILITY;C:\DOS\BATCH
PROMPT=$P$_
```

At the C:\DOS prompt, type SET PROCOMM=D:\TELECOM\PROCOMM\FILES\ and Enter. Type SET and Enter again at the C:\DOS prompt, and specify:

```
COMSPEC=C:\COMMAND.COM
MEMO_DIR=D:\MEMO\
PROCOMM=D:\TELECOM\PROCOMM\FILES\
PATH=C:\;C:\DOS;C:\DOS\UTILITY;C:\DOS\BATCH
PROMPT=$P$_
```

Having sorted that one out, hopefully, let's take a look at some of the other useful things that you can use SET variables for. You saw the hint of one in the environment screen I showed a little earlier in this chapter that included:

```
TODAY=04-28-1990
```

That's one DOS never heard of. Described in greater detail in a later chapter dealing in more depth with batch files, this statement in my system's environment determines when to optimize the hard disk and make backups.

In fact, you can put almost any variable you want in the environment with a SET statement. The date, as shown above, is put there by a batch file that polls the

DOS DATE function from the AUTOEXEC every time the system boots. But how about this one:

```
SET PASSWORD = MUMBO_JUMBO
```

and here it is in the environment:

```
TODAY = 04-28-1990
PROMPT = $P \ $_
COMSPEC = C: \ DOS \ COMMAND.COM
PATH = C: \ DOS;C: \ DOS \ BATCH;C: \ DOS \ UTILITY
PASSWORD = MUMBO_JUMBO
```

You said *SET* and DOS SET it. Again, although DOS doesn't provide any directly, there are much better ways of dealing with security concerns and it has been used here only to demonstrate the flexibility offered by the environment.

The odd couple

Batch files and the environment. This should come as no surprise because the environment is so readily available and any values in it so easily accessed. And while I'll be discussing batch files, as well as the new DOSKEY macros, in more depth in a later chapter, a few thoughts here would seem appropriate.

The possibilities for using the environment are really limited only by your imagination. You can, for example, use the environment to store the date. This is actually a two-stage process, though, because first you've got to *get* the date, and probably the easiest way to do that is with piping and a little redirection:

```
ECHO | MORE | DATE > datefile.bat
```

MORE, in this case, just provides the extra Enter you need to execute this and dump you back out at the DOS prompt. The file I've called *DATEFILE.BAT*, then, looks like this:

```
Current date is Mon 06-04-1990
Enter new date (mm-dd-yy):
```

When you run DATEFILE, however, it does something very interesting. The word *Current* causes another batch file, a one-liner named CURRENT.BAT, to execute the following:

```
SET TODAY = %4
```

The name CURRENT.BAT is important, by the way—this is one filename you can't change and get away with it.

This CURRENT.BAT could do other things as well, but what you're most concerned with here is that it takes the date—the fourth string to the right—as the

variable %4 and puts it in the environment. Type SET at the C: prompt, and:

```
COMSPEC=c:\dos\command.com
TODAY=06-04-1991
PROMPT=$P$_
PATH=C:\;C:\DOS;C:\DOS\UTILITY;C:\DOS\BATCH
```

Once you've got the date ensconced in the environment, there are any number of ways you can put it to work. On the machine most of this book was written on, a 386SX, every time my AUTOEXEC.BAT file runs, it compares its current date to a file that contains the date of the last previous time the machine was run. If those dates aren't the same (==) it initiates a subroutine that optimizes the hard disk, as shown below—something which, if done, should be accomplished before any other programs are loaded:

```
if !%TODAY%==!%4 goto END
\dos\chkdsk
\dos\utility\Vopt c:
\dos\utility\Vopt d:
:END
```

Of course, unfragmenting the hard disk—something Microsoft recommends but doesn't yet support in this release—instantly renders any previous MIRROR copies of the FAT almost useless—certainly the entries for any files that have been moved during the optimizing process. That should be done by the same batch file in most cases.

Using a batch file named CURRENT.BAT is such a neat trick for so many things that it's a shame you can't have more than one CURRENT.BAT file. Ahh, but you can. You can have as many different CURRENT.BATs as you have separate subdirectories to store them in. To get the right one for the job, all you have to do is make the subdirectory with the CURRENT.BAT you want the default directory, and that's the job that gets done.

Note the syntax elements carefully here, though. Each of the values you want to compare are preceded by exclamation points *and* the percent sign used to mark variables (!%), and double equal signs (==) are required to separate the values being equated, as follows:

if !%*environment_variable*%==!%*command_line_variable* goto ...

Note also that, if the variables contain more than one word, those words should be tied together by some delimiter DOS recognizes. Both powerful tools in their own right, batch files and the environment together can open up all kinds of possibilities.

Parent and child

One of the least understood aspects of the environment lies buried in the fact that, to be technical for a moment, *child processes* inherit the environment from their *parent* (DOS). What this says in simple English—and as stated earlier but worthy of repeating—is that every program you load inherits a copy of the environment that's current at the time it's loaded.

You've seen how long PATH statements can actually slow DOS down by making it search laboriously through dead-end after dead-end—the longer the path the more dead-ends. This becomes an increasingly important factor as you move into multitasking environments like Windows or DESQview, where it becomes increasingly tempting just to tack another item on the path . . . and then another . . . and another.

The trick in multitasking is where and when you put things into the environment, and into *which* environment—the parent, sometimes called the *master* environment, or the child. The parent is back at the original DOS level. Every copy mirrors that one—at least until you change it. Then the child takes on its own identity, which mercifully can't be inflicted back on the parent.

Possibly the best way to visualize this is in multitasking. With DESQview or Windows, the environment that's loaded into any window along with a copy of the command interpreter when any application program is loaded clearly has to be a copy. Suppose in one window, though, you load a program like MemoryMate, which has to have a SET statement, as discussed previously. No problem. When you open that window, just make a SET statement before you load the program, with this batch file:

```
SET memo_dir = d:\memo\
c:\dos\memo\memo
```

If you dump MEMO but leave the window open, this is the environment you'll find for *that* window:

```
COMSPEC = C:\COMMAND.COM
TODAY = 04-29-1990
PROMPT = $P$_
PATH = C:\DOS\DV
MEMO_DIR = D:\MEMO\
```

Another window, a big, blank window for experimenting and for loading programs that are rarely used has this environment:

```
TODAY = 04-29-1990
PROMPT = $P$_
```

```
COMSPEC = c: \ dos \ 4dos \ 4dos.com
4DSHELL = /s:de: \  /L:c: \ dos \ 4dos
PATH = C: \ DOS;C: \ DOS \ 4DOS;C: \ DOS \ BATCH;C: \ DOS \ UTILITY
```

This is a different window, a totally different environment (this time even calling up a different command interpreter), and a much longer path. However, if we quit DESQview and go back to the original DOS level, you'll find that the environment there hasn't been changed:

```
COMSPEC = C: \ COMMAND.COM
TODAY = 04-29-1990
PROMPT = $P$_
PATH =
```

It's *still* the one that will get copied, because it's still the parent. So there's a lot to this environment business—an awful lot to squeeze into just 256 total bytes.

Part 3
Productivity

The ultimate measure of any operating system—any software—is what it does for you in terms of productivity. Productivity can be an elusive quality. Poorly configured, DOS 5 (like its predecessors), could be less productive than an earlier release that's used properly.

Just as the times and ways in which your work have changed, so have the issues. This is especially true with today's more powerful machines and vast arrays of memory. Yet, many functions are perpetuated, so it's time to re-evaluate your system needs and how to get the most out of this new release of DOS.

CHAPTER 7

Help, help!!

After years of having cries for DOS help fall on seemingly deaf ears, DOS 5.0 makes help available at every turn through not just one but several separate help facilities, three unique to specific features of this new release and one all-purpose facility that, taken together, will save most of those trips to dig out the documentation:

DOSHELP.HLP
DOSSHELL.HLP
EDIT.HLP
QBASIC.HLP

We will look at the specific help facilities DOS now has for DOSSHELL, EDIT and QBASIC when we look in detail at those functions. They are important here however to call attention to the extent of help available overall in this release.

At the raw DOS command line level release 5.0 introduces an all-new on-screen HELP facility, accessible either by typing the word HELP followed by the command (HELP *command*) or simply typing the command followed by the help switch (/?). Old DOS-hands and newcomers alike should find the new DOS HELP facility a valuable resource; new users as a training aid and even power users at least to refresh the gray matter when it comes to some of the lesser-used commands and syntax—and especially for some of the new commands or where new options have been added for some old ones.

The extent of the on-screen HELP offered by 5.0 varies greatly, ranging in some cases from a single line with simple commands that support support no switches to others that offer almost a screenful. HELP is essentially limited to the information presented at the beginning of the manual listings for each program: the basic command and syntax for options, a brief description of any switches that

are supported or special parameters that are supported and etc. Where DOS offers several commands with overlapping but differing features—COPY, XCOPY, and REPLACE for example—such other possible alternates are not referenced.

On-screen help as implemented in DOS 5.0 seems at first glance to be multi-layered. And in fact it is, although the separation of the layers is a little different than you might suspect.

All of the substantive HELP for DOS 5.0 commands is intrinsic to the commands themselves and is completely independent of any other system resources. Notable exceptions are device drivers and commands like SHELL and INSTALL, that are used only in the CONFIG.SYS

If the executable file for the command itself provides not only whatever HELP text you see on screen but also a means of accessing it without external assistance, no HELP exists as far as DOS is concerned. This may at a glance seem contradictory to something we will look at just a little farther down. It really isn't though if you'll bear with me for a moment.

Figure 7-1, for example, shows how a little benign poking around inside the DOS 5.0 XCOPY.EXE file reveals this ASCII text. Granted, in its native form it doesn't make a lot of sense. But Fig. 7-2 shows what happens when you issue an XCOPY command with /? on the command line (XCOPY /?) and let DOS sort it out for you.

If you compare the dump in Fig. 7-2 with the actual HELP screen that's displayed, you'll see it's all there in the raw dump; DOS just makes it look prettier for you on the screen. Another case in point—and this one is a beauty—is COMMAND.COM. Shown in Fig. 7-3 is just a partial DUMP of all the ASCII help that's buried in COMMAND.COM in release 5.0 (over 10K in all), but if you look carefully you can find the same text DOS displays for any of several COMMAND.COM internal commands, if entered with the /? switch.

In the above, you accessed HELP by entering *command* /? on the command line. Now why don't you try something different? Using one of the same com-

```
001080      Copies files and directory trees.NXCOPY pathname1 [path
0010C0   name2] [[/A]|[/M]] [/D:date] [/S [/E]] [/P] [/W] [/V]  path
001100   name1 File(s) to copy.1 pathname2 Location and/or name of n
001140   ew files.;  /A Copies files with the archive attribute s
001180   et,-/ doesn't change the attribute.;  /M$ Copies f
0011C0   iles with the archive attribute set,-/ clears the archive
001200    attribute.D /D:date  Copies files modified on or after the
001240    specified date.8  /Si Copies subdirectories, except empt
001280   y ones.D  /E0 Copies empty subdirectories along with occu
0012C0   pied ones.5  /P< Prompts before creating each new file.
001300   ;  /W- Waits for you to press a key before copying.&  /V3
001340    Verifies each new file.
```

7-1 A look inside XCOPY reveals that the data for the new MS-DOS HELP facility is actually contained in the program file rather than in a separate HELP file.

```
Copies files and directory trees.

XCOPY pathname1 [pathname2] [[/A]|[/M]] [/D:date] [/S [/E]] [/P] [/W] [/V]

  pathname1  File(s) to copy.
  pathname2  Location and/or name of new files.
  /A         Copies files with the archive attribute set,
              doesn't change the attribute.
  /M         Copies files with the archive attribute set,
              clears the archive attribute.
  /D:date    Copies files modified on or after the specified date.
  /S         Copies subdirectories, except empty ones.
  /E         Copies empty subdirectories along with occupied ones.
  /P         Prompts before creating each new file.
  /W         Waits for you to press a key before copying.
  /V         Verifies each new file.
```

7-2 The help screen displayed for XCOPY.

```
009340    splays or changes the system time.TIME [time]{TIME with
009380    no parameters displays the current time setting andprompts for
0093C0     a new one. Press ENTER to keep the same time.8Displays the c
009400    ontents of a text file.TYPE pathname%Displays the MS-DOS v
009440    ersion.VERTurns disk-write verification on or off.VER
009480    IFY [ON|OFF]To display the current VERIFY setting, enter VER
0094C0    IFY without a parameter.ADisplays a disk volume label and seri
009500    al number.VOL [drive:]ACalls one batch file from another.
009540    CALL pathname [arguments]=Records comments (remarks) in bat
009580    ch files.REM [comment]8Suspends execution of a batch file.
0095C0    PAUSE [message]TEchos messages to the display, or turns co
009600    mmand echoing on or off.ECHO messageYECHO [ON|OFF]ECHO
009640     without any parameters displays the current command echo settin
009680    g.BDirects batch file execution to a labelled line.GOTO la
0096C0    bel  label   A text string used elsewhere in the batch file
009700     as a label.Labels appear at the beginning of a line, preced
```

7-3 A dump of ASCII text found inside COMMAND.COM, revealing help text for commands supported internally.

mands you looked at earlier, enter HELP *command*, instead. Thus, the command:

```
HELP XCOPY
```

Although this time you've invoked HELP.EXE, one of DOS 5.0's new external commands, what you see displayed is *exactly* what the /? switch showed you (in Fig. 7-2). And for good reason—all HELP.EXE really did behind the scenes was run XCOPY /?. It neither added nor subtracted anything—in fact, this is about the only help text in HELP.EXE, just the raw ASCII makings for the help and error messages.

Why, then, do we need HELP.EXE? Good question. Run by itself (with no other command specified as an argument), HELP does provide you with a capsule roundup of all of the DOS 5.0 commands that support on-screen help, as demonstrated by the help screen in Fig. 7-4—note that the program pauses before displaying the successive screens.

```
For more information on a specific command, type HELP command-name.

APPEND   Allows applications to open data files in specified
         directories as if they were in the current directory.
ASSIGN   Assigns a different drive letter to an existing drive.
ATTRIB   Displays or changes file attributes.
BACKUP   Backs up one or more files from one disk to another.
BREAK    Turns extra CTRL+C checking on or off.
CALL     Calls one batch file from another.
CD       Displays the name of or changes the current directory.
CHCP     Displays or changes the active code page number.
CHDIR    Displays the name of or changes the current directory.
CHKDSK   Checks a disk and displays a status report.
CLEANUP  Cleans up left-over files after MS-DOS 5.0 installation.
CLS      Clears the screen.
COMMAND  Starts a new MS-DOS command processor.
COMP     Compares the contents of two files or sets of files.
COPY     Copies one or more files.
CTTY     Changes the terminal device used to control your system.
DATE     Displays or sets the date.
DATECHEK Returns ERRORLEVEL corresponding to day of week
DEBUG    Starts Debug, a program testing and editing tool.
DEL      Deletes one or more files.

---More---
```

7-4 If the HELP command is given without specifying a command name, a command list with abbreviated help is displayed.

The data for these screens comes from the file DOSHELP.HLP, which HELP.EXE accesses when given no specific command to search for. However, if you look carefully, you'll find an entry included in the screen that's not a DOS 5.0 command.

DOSHELP.HLP is, in fact, nothing but an ASCII text file and, as such, can be edited either with the new DOS 5 EDIT utility or with any word processor that produces straight ASCII files. HELP.EXE and DOSHELP.HLP together can then be used to provide at least a rudimentary level of help for *any* command—including other than commands specific to DOS 5.0—completely independent of any other DOS commands or features.

Bear in mind, however, that no, I repeat, *no* help information in the DOSHELP.HLP file, whether information supplied in the original file or added later, can be accessed by the HELP command, other than as a display of the entire DOSHELP.HLP file, as shown in Fig. 7-5. Here, even though you've been able to add a line for a program called DATECHEK, a request through HELP for any information specific to this command (or any other unsupported command) returns:

```
Help not available for this command
```

HELP.EXE can and will, however, display any help provided by other than DOS programs that provides help text and supports the /? switch as a legitimate means of access to that help.

But then, if a program supports the /? mechanism for accessing embedded help text, you really don't need HELP.EXE, do you? You need HELP.EXE only if you want or need to display the contents of the DOSHELP.HLP file—and if you really want to know, you don't even really need it then. Instead of an 11,443-byte .EXE file, you can do the same things with the following fairly simple batch file I call HELP.BAT (although it's still going to take up 2048 bytes of hard disk space to store it):

```
@ECHO OFF
IF !%1 = =! MORE < \DOS\DOSHELP.HLP
IF NOT !%1 = =! %1 /?
IF ERRORLEVEL 2 ECHO Help file could not be found
IF ERRORLEVEL 3 ECHO Help not available for this command
```

I've even managed to include two ERRORLEVEL tests to report specific errors that might be encountered during execution—two of the same messages HELP.EXE can respond with when appropriate. The list of error messages could probably be expanded with the corresponding ERRORLEVEL returns for various conditions determined experimentally if not documented. At the time this was written, the HELP command wasn't fully documented and the values used here were determined experimentally.

As with several other batch files demonstrated elsewhere in this book, HELP .BAT uses the exact same syntax as HELP.EXE, displaying the contents of the DOSHELP.HLP file if no other command filename is specified or if one is specified executing that command with the /? switch.

Line length limitations make it impossible to include all the features shown in the HELP batch file in a DOSKEY macro, but the basic functionality (excluding ERRORLEVEL testing) can be duplicated with a macro similar to the one below:

```
DOSKEY HELP=IF !$1 = =! MORE $L\DOS\DOSHELP.HLP$TIF NOT
!$1 = =! $1 /?
```

Normally, the DOSHELP.HLP file contains a number of lines of text that aren't displayed by HELP.EXE; instructions for how to add other programs to those listed. To prepare DOSHELP.HLP so our batch file or macro does not display them either is simply a matter of going in with a word processor (or the new DOS EDIT feature) and stripping out everything but the text you want displayed, as in the excerpt shown in Fig. 7-5 (the material in italic can be deleted).

A better solution would be to copy the stripped-down text under a new filename—JIMSHELP or almost any new filename you might care to use (and

```
@ for a command description should be preceded by white space.  Commands
@ must be added in alphabetical order.
@
@ Copyright 1990 Microsoft Corp.  All right reserved.

APPEND    Allows applications to open data files in specified
          directories as if they were in the current directory.
ASSIGN    Assigns a different drive letter to an existing drive.
ATTRIB    Displays or changes file attributes.
BACKUP    Backs up one or more files from one disk to another.
```

7-5 When editing the general abbreviated help file, items shown in italic can be deleted, leaving only the actual command list and help.

change the name of the file input to MORE in the batch file or macro)—and keep the original intact.

This isn't to say you should run right to your keyboard and erase HELP.EXE and replace it with a batch file. I didn't—I just had fun playing with it. Hopefully, however, it will shed a little light on how the new DOS HELP functions operate. And what's most important here is that on-screen HELP is now available, and a little help never hurt anybody.

CHAPTER 8

Fact and fiction

The ultimate measure of DOS or any operating system is what it does for you—or, more correctly, what *you* can do with *it*. The name of the game is productivity, and to put 5.0 in its proper perspective, you need to look at some of its various features in that light. DOS 5.0 is good. I've said that before and I'll say it again. But simply installing DOS 5 doesn't guarantee it's going to do the job it should for you. You have to update your thinking as well as your DOS.

This is especially important because DOS continues to support so many antiquated features, some dating back as far as version 1.0. The mere fact that they're still in DOS 5, however, doesn't mean they're still viable. Many have, in fact, been superseded by new and better functions that either directly or at least tangentially perform the same—or in some cases, overlapping—functions. In at least one case, a function still supported in 5.0 (INSTALL) actually seems to run counter to the DOS 5 philosophy of loading as much as possible above 640K to free more conventional memory for use by applications.

The reason for continued support of functions and features that are no longer valid is a direct result of Microsoft's philosophy. The idea is that users of earlier DOS versions can upgrade without really having to relearn new commands or syntax for familiar operations. This is why the FORMAT command, for example, still supports all the clumsy old conglomeration of switches that accumulated through the years, but also now supports the new form factor switch, which really makes the old ones obsolete. Worthy in principle, it's this kind of inconsistency, however, that many users find so frustrating.

In this chapter, then, I'll examine some of DOS's better productivity tools, some real dinosaurs, and, hopefully, dispel a myth or two along the way.

Shell game

In terms of productivity, the new DOSSHELL will certainly be of interest to many users. While I've devoted an entire chapter to it elsewhere in this book, I think a moment to look at it in terms of productivity is warranted, especially because productivity is ultimately the whole purpose of a shell.

I suspect that users will quickly divide into two camps on the shell—some will love it and others will hate it, with most experienced DOS users falling in the latter category. Inexperienced users generally appreciate any help they can get, and any help at all to them is valuable. What constitutes help and productivity have different meanings to different levels of users, however, so while users new to DOS might be helped, overall I think experienced users will find it actually slows them down—if they even bother to try it.

Experience level aside, the new task-swapping feature should interest anyone at any level who isn't already using one of the several 3rd-party multitasking or context-switching packages that are available. Let's face it—*anything* that allows you to switch back and forth between two or more applications or just put everything on hold while you pop out to DOS to do something you can't do easily (if at all) from within the shell is to the good.

Any time you have to close your file and exit a program, even just to get a peek at something in another file—costs you time. And then more time when you try to find your place again when you go back and have to start all over. For myself, I couldn't live without, at very least, some means of switching quickly back and forth between sometimes as many as a half a dozen tasks.

Given that, as a productivity tool, the new task swapper adds a new dimension to DOS. That's the good news. The bad news is that the "new" DOSSHELL task swapper is literally years behind, not only in how it's being added to DOS but in the way it's implemented. Even some of the earlier versions of Software Carousel, for example, ran rings around it, and in these days of multitasking . . . let it suffice to say that Bill Gates isn't giving you anything here that could impact on Windows sales. DESQview is certainly unconcerned and I heard even a snicker from the guys at Softlogic Solutions (Carousel).

The only other productivity issue of significant interest to the broader user market is in the area of file management, and even the glimpse we saw in DOS 4 was, overall, a marked improvement over anything we'd seen in DOS before—with its down-only scrolling and totally noninteractive TREE and DIR functions. Again, even acknowledging the fact that much of this was introduced with DOS 4.0, DOS is years behind in offering something that's sparked the development of a whole 3rd-party software industry topped by products like XTREE.

About the best thing I can say about either of these areas is that the DOSSHELL doesn't cost you anything—it's free with DOS 5.

A brave new world for DOS

Certainly the most significant advance in this release is in the area of memory management—of special interest to anyone currently using or even thinking of purchasing an 80286 or higher machine. Microsoft has defined extended memory as the future direction for the industry, and the ability of the new DOS to move most of itself out of your way, freeing up some 47K of extra conventional memory for you to use in any way you see fit, is a major change.

Here again, because this subject is so important, I've devoted an entire chapter to it. But certainly some observations are warranted here because, in this, you not only have the key to greater productivity, but to the future—to the very survival—of DOS itself as a viable operating system long after many pundits had predicted its demise.

DOS can never be more than a one-megabyte operating system—an anachronism in this age of gigabytes. It's a one-megabyte system because of short-sighted decisions that were made (reportedly even after warnings from Intel and probably others) about a generation of computer chips that would quickly make the concepts DOS is founded on obsolete—as indeed they have. Still, thanks to a lot of ingenuity—mostly by 3rd-party developers—enough ways have been found to get around DOS's limitations to not only keep DOS viable, but take much of the steam out of the steamroller that was supposed to deliver OS/*who*.

And DOS now offers add-on memory management utilities that support the better of the work-arounds that I've seen to date, not only for your convenience but, in a twist of irony, to meet even the most basic need of Microsoft's own Windows 3.0. Here, although by best estimates at least two years behind the demonstrated state of memory management technology, HIMEM.SYS, DOS's new XMS extended memory driver and its companion 386/i486-only EMS emulator, EMM386.EXE, are powerful tools. While the XMS driver currently supplied by Microsoft with Windows 3.0 is also named HIMEM.SYS, at the time this book was written that version was a far more rudimentary device and it was recommended by Microsoft that users of Windows 3.0 running under DOS 5.0 replace the Windows version with the later and significantly more powerful DOS 5.0 version. DOS's HIMEM.SYS and EMM386.EXE are capable of delivering performance equalling if not exceeding the current needs of probably the majority of users.

Although these two device drivers lack some of the sophisticated features of some 3rd-party drivers, up to their design limits they have the capability of doing as effective a job of memory management beyond 640K as any I've seen. I have, in fact, been pleasantly surprised by what I've been able to accomplish with them in an area of special interest to me. They aren't shabby.

What they lack most, in typical DOS fashion, is the kind of helpful user inter-

face, even automated installation, that's provided by any of the better 3rd-party memory managers. In the faint praise department, MEM, the only new external command added in DOS 4.0, has been enhanced and is a useful adjunct, particularly when working with 386 and i486 machines. It still leaves the user to do the real work, however, when it comes to loading device drivers and TSRs above 640K—which, beyond the entry level, can quickly escalate into a lot of time-consuming trial and error effort.

But if you've never worked beyond 640K and you've got a 286 or higher system, now is the time to jump right in and get your feet wet. DOS 5 will surely do that for you, and might well have all the memory management you need.

Speed demons

We all know that any time you can use RAM instead of doing disk reads, you can expect to speed things up significantly. Generally, that's a fact, but note the operative word is *generally*. This is one of those places where a little knowledge can get you into trouble, actually slowing down your system rather than improving productivity. In other cases, you can chew up precious RAM in ways that don't slow you down but certainly don't help you that much either.

Two ways you can boost productivity are through disk caching and the use of RAM disks, assuming you have enough memory to support them. Of those, disk caching is probably the most sure-fire. RAM drives can be very helpful, too, but require more savvy to be used effectively.

SMARTDRV.SYS

Disk caching isn't something that's going to reduce all access times to RAM speed, and it's not something you can quantify down to the point of saying "Wow! I saved a whole six minutes this afternoon." But on an average it generally does speed most disk operations—the slower the hard disk, the more significant the saving.

Some programs keep going back to read the disk, often simply rereading data they've read quite recently. What disk caching does is save a copy of the most recently read or written disk sectors in RAM so, if called for again, they can be accessed from the cache in RAM.

On writes, a good cache also acts as a buffer, taking in the data to be written at RAM speed, then turning the system back to you while it performs the actual writes. You can often see the affect here—the drive light indicating the drive is still being accessed well after control of the system is turned back to you.

In typical DOS fashion, SMARTDRV doesn't give you any indication just exactly what it's doing for you, but I have another cache that does. During a typical work session, I was given the information shown in Fig. 8-1. You just know this has to translate into a significant amount of time saved.

```
63,446 logical transfers.
10,681 physical transfers.
52,765 transfers saved.
    83 percent saved.
```

8-1 Report indicating the percentage of hard disk access disk caching saved during a typical work session.

To achieve this kind of performance, however, takes more than just a little bit of RAM. It goes without saying that the amount of data a cache can hold is a function of the size of the cache. To a point, there's a fairly linear increase in performance as cache size increased, but only to a point. Beyond that, the law of diminishing returns seems to take over.

For my work, somewhere between 500K and 1Mb seems to do the most effective job—the session in Fig. 8-1 was with the cache size set to about 500K. At one time, that was a pretty costly item, but, at today's prices, it's only about $35 worth of RAM, a difference that turns costly into cost-effective.

Sure, you *can* cache with as little as 128K—the SMARTDRV default is 256K—but I don't recommend it. If you're going to do it, use at least the default value, and if possible a little more.

SMARTDRV.SYS can use either extended or expanded memory, making it suitable for use on *any* DOS machine from 8088s on up. However, because DOS provides an EMS driver, except for 80386 (and up) machines, a 3rd-party memory manager is required for use with 8088s or even 286s if you should opt to use expanded memory rather than extended memory.

If you have a 286 or higher with extended memory, SMARTDRV will probably perform a little better in extended memory. At this time, however, you shouldn't attempt to load SMARTDRV.SYS into upper memory blocks on 386 or I486 machines. This prohibition doesn't apply to some 3rd-party disk cache programs, but if Microsoft says don't do it, I wouldn't recommend it.

A superfluous redundancy

If you're using disk caching—any disk caching—you want to take a hard look at the number of buffers you're using. On most machines today—anything with 512K or more RAM—DOS 5 defaults to 15 buffers. DOS buffers themselves are small, rather primitive caches—each buffer is a tiny cache just big enough to hold the contents of one disk sector (512 bytes); the more buffers you have, the more sectors.

There's more at stake, however, than using memory on what might seem like "more of a good thing." With buffers, unlike a full-blown cache, not only is more not a good thing, but it can, in fact, be detrimental, actually slowing down your system.

The problem with buffers is this—when a program requests some data that's

stored on disk, DOS first checks to see where on the disk the data is, or what sector it's stored in. But DOS doesn't go straight to the disk at that point. Before it does, DOS checks its buffers. If it finds the data there, fine. If not, it reads it from the disk.

It's the "if not" that's crucial here, because polling all these little independent single-sector buffers, checking to see if any of them have the data DOS is looking for, takes time. More buffers, more time. A well designed cache, on the other hand, operates much the same way as an indexed database, only having to check its index to know where and what to do.

Even with disk caching, you *do* need buffers—some at least, but far fewer than the number DOS will give you as a default value in most cases. About five seems to be a fair number.

Unfortunately, having a properly small number of buffers is something that the installation programs for many applications packages fail to recognize or take into account. And because they don't, you'll sometimes have to change your CONFIG.SYS just to be able to install certain programs. Also unfortunately—at that point, however—after the new software is installed, a lot of unwary users leave those extra buffers in there, unaware that they really aren't needed. The number of extra buffers called for is usually small enough not to cause a noticeable degradation of system performance, but like any excess baggage, you're better off without them.

Ideally, then, in most cases any extra buffers that somebody's installation program insists you have before it will install should probably be removed again as soon as the new software is installed.

FASTOPEN

This is one you can pretty much forget about, in my opinion, especially if you use disk caching. Even without caching, I've never been able to see that much real benefit and, once you introduce a cache like SMARTDRV to your system, any benefit might seem to disappear completely.

FASTOPEN is also a form of caching, but differs conceptually from either SMARTDRIVE.SYS or buffers. Instead of caching data, this one caches only paths and filenames, speeding the relocating of recently accessed files.

On the plus side, the memory requirements for FASTOPEN are very modest, particularly compared to full caching. It requires only about 4K to load the program, and you can see the number of recent filenames it can remember, each one you make provision for requiring only 48 bytes or about one kilobyte for 20.

At one time, the pre-DOS 5 SELECT (installation) program would automatically install FASTOPEN whether you wanted it or not. DOS 5 doesn't even try—somebody must have gotten the word—but it's still there if you really want to use it.

RAMDRIVE.SYS

The term *disk* is used rather loosely in this business. Even when most people talk about a *hard disk*, they're not really talking about the physical spinning platters that are hermetically sealed inside a box, but rather the illusion created by DOS—an *illusion* of having one or more disks based on artificially partitioning the data storage space.

A disk, to DOS, is nothing but some form of data storage accessed in a particular way. Accessed in that accepted manner, a properly configured block of RAM becomes a dandy disk that can literally run rings around anything that whirls and spins.

The time saved by such a virtual disk is probably comparable to the time saved by caching with a large enough cache, but in this case it can be quantified. I compared the times required to write an 80K file to different "disk" media. For the RAM drive, it took 1.4 seconds; for the hard drive, 3.4 seconds; and for a 720K floppy, 19.2 seconds.

Clearly, the time saved over conventional disks is a function of the type of disk in question, and even within the hard and floppy disk categories there are many variations. These times, however, are at least sufficient to warrant a closer look.

You can, for the sake of argument, assume that the time saved by caching would be comparable to that saved were you to use a RAMDRIVE. Unfortunately, trying to compare a RAM disk with disk caching is a lot like trying to compare apples and oranges, but bear with me.

Unlike a cache that only becomes really effective in the neighborhood of 256K and up, size has no effect on any time saving achieved through the use of a RAM drive, as long as the size of the "disk" is large enough to hold complete files. Like any disk, you can't load half of a file on one, and the rest someplace else. (A cache isn't concerned with files, but only with sectors which might and often do represent only fragmented parts of files.) Bigger, of course, is always an advantage, but you'd be surprised what can be done even with, say, a 64K RAM drive—which happens to be the default size.

How much is enough?

While individual needs will vary as much or more than individual resources, even if you have just the extra 384K that's left over on a typical machine shipped with one megabyte installed, that in itself is enough for a credible SMARTDRV cache, plus a minimal RAM drive. In fact, even on a 286 or any machine that will support extended memory, out of 384K you can have a High Memory Area that allows you to pop about 47K—most of DOS itself—up above 1024K, *plus* a 256K SMARTDRV and still a small but usable RAM drive. Or a little bigger RAM drive and a little smaller cache, depending on your likes and needs.

Granted, these numbers aren't going to deliver performance that will blow you out of your socks, but the point is that even with this small amount of memory—memory many users waste entirely simply for lack of an XMS driver such as DOS 5's HIMEM.SYS—you can get a pretty good jump on speeding up your system.

Where you go with it from there is between you and your checkbook. There are several ways a RAM drive can be used to speed your operations, however, and the more you know about using RAM disks the more you're likely to want to build a bigger one.

One of the easiest ways a RAM disk can speed things up for you is when you use one in conjunction with any program that creates temporary files for any reason. My word processor does it whenever it prints a file and some DOS functions do it. And, like any save-to-disk operation, saving to a RAM disk saves you time. Sometimes this is done by setting TEMP or TMP as variables in the environment, like this:

```
COMSPEC=E:\COMMAND.COM
TEMP=e:\
TMP=e:\
TODAY=05-08-1991
PATH=C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY
PROMPT=$P$_
DIRCMD=/o/p
```

Here I've used both because some, mostly older, programs use TMP rather than the preferred TEMP. This way I have my bases covered. Sometimes the programs themselves will provide the mechanism for specifying where whatever constitutes temporary files get dumped. In any case, the operative word is *dumped* because they are, by definition, of a temporary nature, which is what RAM drives deal with best.

Aside from speeding things up, a fringe benefit of going to a RAM disk rather than your hard disk with temporary files is that, by reducing hard disk use, you reduce fragmentation—which affects not just temporary files, but any files.

Another way you can use a RAMDRIVE to speed things up is by copying your command interpreter to it, then resetting the COMSPEC= so it goes to the RAM disk rather than the real disk every time a program has to load a secondary command processor—in case you missed it, that's where the COMSPEC in the environment shown above says to look. My AUTOEXEC.BAT does that automatically.

These are only a couple of suggestions, most of which are the ways I use a RAM disk every day. Here again, although RAM disks have been around since long before PC DOS first introduced VDISK.SYS in version 3.0, at today's RAM prices they're a better bet than ever.

As a last thought, keep in mind that RAMDRIVE.SYS, while similar in func-

tion to the VDISK.SYS which is furnished with PC DOS, is slightly different in some respects, although in most cases these differences won't be noticeable to the average user.

The wrong way around with APPEND

Intended as a productivity tool, APPEND goes about the job all wrong. And it's too bad, because if it weren't backward it could really be a help.

The idea behind APPEND is to provide a path for data files in much the same way the PATH accesses executables. And it does. You can access any files anywhere in the APPEND path as easily as if they were in the default directory. But if you make any revisions to those files, the revised files are saved to the default directory, regardless of where you have and want to keep them.

There are a couple of ways of getting around this, but, frankly, none that seem to be worth the effort. Not when there are better tools so close at hand.

SUBST: the shortest route

One of my pet DOS productivity tools is SUBST. Hardly new—it's been around since DOS 3—but a lot of users have never picked up on it and I don't know why. The more thoroughly you organize your files into well ordered directories and subdirectories, the more you really need this one—I don't know how I'd ever get along without it.

With SUBST, you simply *substitute* a nonexistent drive for a subdirectory name—or a whole string of subdirectories. Whenever I load my word processor (from a batch file), it sets up this list of substitutions, speeding access to all of my most-used files:

```
\dos\subst i:   \files\tab\dos-more\book\figures
\dos\subst j:   \files\projects
\dos\subst l:   \files\letters\1991
\dos\subst m:   \files\magazine\guide
\dos\subst o:   \files\tab\dos-more\book\finished
\dos\subst p:   \files\personal
\dos\subst q:   \files\letters\1991\queries
\dos\subst r:   \files\tab\beyondos\revise
\dos\subst s:   \files\tab\dos-more\book
\dos\subst v:   \files\tab\dos5
\dos\subst w:   \files\projects\33
\dos\subst x:   \files\fiction\misc\short
```

To call up something in \FILES\TAB\BEYONDOS\REVISE subdirectory, instead of typing \FILES\TAB\BEYONDOS\REVISE*filename*, I just type R:*filename*. This works even where the substituted directories have subdirec-

tories. First specify the substituted drive designation, then the subdirectory—just as if it were a real drive—and then the filename. This saves me 25 keystrokes every time I go to that directory, and hundreds, probably thousands of keystrokes daily—not even including the number of times I'd have to correct all the typos in all those extra keystrokes.

Batch file bingo

As thoroughly as I'll explore batch files and macros in the next chapter and elsewhere in this book, I'm not going to belabor them here, except as another way around the path problem for data files.

If you keep all your data files for a particular program in one directory, one of the easiest ways of making them easily available is to make whatever *drive**directory* they're in the default while you're working with that program. You can do that from the command line of course, but it's easier from a batch file, like this one:

```
CD \FILES\SUPRCALC\CURRENT
\DOS\SUPRCALC
```

which will change the default directory and calls up the program—in this case, a spreadsheet program—*from* that directory. Unlike APPEND, the logic is reversed. But it works, and in the final analysis that's what matters most.

Past their prime

As you can see from some of the things discussed in this chapter, productivity is not something that lends itself to easy answers, nor is there any single strategy that guarantees it. Rather, it's generally a composite that encompasses a number of factors. What's best for you might well be different than what works for me. At least this is a starting point.

A lot of things in DOS are throwbacks to a different time. ASSIGN and JOIN and EDLIN, just to pick on three that come to mind immediately—are three that you can probably erase, forget, and be done with now and forever. There are others that will probably always be around, though, like BREAK and VERIFY—the latter of which is guaranteed to slow you down while proving nothing. And there are more. DOS doesn't seem to have the heart to throw them out.

This is a new DOS and the beginning of a new DOS era, however, and maybe it's time for you to re-evaluate your system, especially if you've been using DOS for quite a while, possibly even going back to when these were the best DOS had for us to work with.

CHAPTER 9

Batch files and DOSKEY macros

I'm always surprised by the number of DOS users who still don't use batch files other than their AUTOEXEC.BAT and CONFIG.SYS—and even those only because they have to. I'm not surprised, however, that many of these people are the same ones who think DOS is so hard to work with. If earlier DOS releases had any civilizing virtue, it most surely was the batch file. Batch files have traditionally provided a cheap-and-dirty way of automating repetitive tasks.

And now, beginning with 5.0, DOS has added a new macro function with its new DOSKEY utility. DOSKEY, while roughly paralleling the batch file, has several advantages over batch files—but at the same time certain limitations. Because they are so similar in purpose, despite their differences, I will, for the most part, address batch files and DOSKEY macros together in this chapter. This, hopefully, will accomplish several things:

- Demonstrate to old DOS hounds how easily they too can learn new tricks.
- Provide a side-by-side comparison pointing up the relative advantages of one over the other in specific situations.
- Save me a lot of extra typing.

While whole books have been devoted just to batch files, and the new DOSKEY macros will certainly provide more grist for the mill, I hope that this will provide enough to get you up and running anyway.

At the most elemental level

Anyone who can deal with DOS at the command-line level at all can create useful batch files or macros. It's that easy because that's all batch files or macros really are—a sequence of DOS commands or even just a single DOS command, with parameters you have to look up *once*. That, in fact, is a perfect way to get started in batch files if you're not already doing them.

Say you've got some DOS command you want to use repeatedly, but you can't remember the syntax and the parameters. Sit down in front of your computer with the documentation, play with the syntax until you get it right, then save the command name to a file with a .BAT extension tacked onto whatever name you want to give it, and you've got a simple batch file.

Other batch files or macros—but especially batch files—can perform more complicated tasks, in many cases even using simple built-in logic statements to branch to subroutines if certain conditions are encountered. Properly used batch files (or macros to a somewhat lesser extent) can report errors and even abort processes in certain circumstances.

Many power users tend to look down their nose at the lowly batch file—they use them, but all the while decrying how limited they are in what they can accomplish. DOSKEY macros will no doubt get their share of criticism, too.

No one can argue the fact that DOS's limited batch file language—and the even more limited macro capabilities—fall far short of what many people would like to see, particularly as evidenced by some of the better 3rd-party batch file enhancement programs available out there. Yet I'm also amused sometimes by some of the "enhancements" that, when analyzed more closely, in fact do nothing you can't do with ordinary DOS batch file commands and just a little understanding. This, of course, isn't always the case, but unquestionably much of whatever power DOS's batch file command set has is wasted on too many users.

Most users probably got their first taste of batch files when they had to write or modify either their AUTOEXEC.BAT or CONFIG.SYS file—while the CONFIG.SYS is hardly typical, it's really just a special kind of batch file.

The AUTOEXEC.BAT is a typical batch file in almost every way and, while usually longer than most other batch files, is an excellent model to look to. Here, for instance, are three lines from my AUTOEXEC.BAT, which have one important thing in common:

```
PATH C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY
MODE COM1:9600,N,8,1,P
PROMPT $P$_
```

Something in common? Yes. All of the three commands in the file—*any* of them, for that matter, could be removed from the AUTOEXEC.BAT and punched in

manually and independently at the DOS command line. The most important thing about them is that there's absolutely nothing unique about any of them. They are just simple DOS commands that are listed, one after another, in a file.

This is true of *any* batch file (or macro) command. With the exception of the GOTO or SHIFT commands, a valid batch file command line must be something that can be input manually and executed from the DOS prompt as a stand-alone command. Granted, many of the lines used in batch files would make little or no sense as stand-alones. But the point is that, to be valid in a batch file or a macro, they *must* be lines that will at least run without returning a *Bad command or filename* message.

This should not only help to clear away some of the mystique, but also provide you with a marvelous batch file and macro development tool—the DOS prompt. A lot of times, in fact, I find myself experimenting at the DOS prompt with commands I hope to include in batch files or macros. This is particularly true when working with conditional statements where I might have to make a choice between two seemingly valid ways of doing what I want to do: IF *this* or IF NOT *that*, for example.

The actual line I work with from the DOS prompt is not always the same exact line I'll put in the finished batch file. Often, just to test the logic, I'll try something like:

```
IF EXIST some_condition ECHO It works!
```

I'll examine conditionals in greater detail later in this chapter, but for now, once you've got your echos echoing, you can put in the GOTOs and whatever else you need to make a real working batch file or macro. Particularly for beginners, this technique of stepping through a would-be batch file or macro line-by-line isn't a bad idea.

A language of their own

In addition to being able to include any file with a .COM or .EXE extension for execution, DOS provides a set of functions intended almost exclusively for batch file use—most now also suitable for use in DOSKEY macros, as well. Taken alphabetically rather than in order of importance or frequency of use, these include:

CALL	IF [NOT]
DO	IN
ECHO	PAUSE
FOR	REM
GOTO	SHIFT

In addition to these commands, you also have EXIST and ERRORLEVEL, which

are essentially unique to batch file and macro usage. They're really not commands *per se*, however, but are instead conditional tests that are used most often in IF and IF NOT statements. I'll look at all of these individually, not only as they apply to batch files but also to the extent that they're suitable for macro use (because there *are* differences).

Symbols with special meanings

While ultimately a command run by a batch file or DOSKEY macro must emulate the command(s) as run directly from the command line (with a few exceptions), DOS provides you with several valuable tricks. Paramount among them is the ability to enter strings or parameters on the command line that will be passed through to specific commands within the body of a batch file.

The special symbols recognized by batch files differ from those recognized by DOSKEY macros, making it impossible to simply load and run old batch files now as DOSKEY macros. There are, however, at least functional equivalents in most (but not all) cases.

: (used to preface labels in batch files only)

The colon is always used to mark the beginning of a label that marks the beginning of a branch or subroutine. You've seen it used in several instances already. A label can be just a single character or almost any string of legal characters you want to make it. That excludes any characters that have special meanings to DOS, but any combination of letters or numbers should work.

I generally prefer at least three characters for labels; preferably characters that spell out an appropriate word like END, NEXT, PROBLEM, or whatever. It doesn't matter to DOS, and by using recognizable words it makes it easier to read down to through a batch file later if you want to change it.

% (used in batch files only)

This is probably the most common usage of batch file and macro variable—numerical variables (%0 through %9), which are used to indicate command line position from left to right, as shown here:

```
COMMAND string1 string2 string3 string4 string5 string6 string7
string8 string9
```

with *string1* corresponding to %0, *string2* corresponding to %1, etc. When the batch file (*COMMAND*) runs, then any strings entered on the command line following the batch file are passed through to any commands within the batch file that request them. The following, for example,

```
batch_name string1 string2 string3
```

will pass the parameters this way:

```
DOS_command1 %1 %3
DOS_command2 %2 %3
```

0%, as you can see, represents the first string, which is generally the command itself, making it of little value as a variable. As a practical matter, that effectively limits you to %1 through %9 then, but for most purposes that's more than sufficient—and a greater number can be used by employing the SHIFT command.

Something you must watch out for when using this technique, however, is to be certain that, when the command line data is entered, there will always be a *string2* before you type the data you want passed through as *string3*. I stress this because, after passing *string1*, DOS is going to treat the next string that you enter as *string2*, whether that's what you intended or not.

In an example like this, you couldn't sometimes have a *string2* and sometimes not, and still hope to pass a *string3* to the second command. If you omit *string2* as a parameter for *command1*, but then go on to type parameters intended for *command2*, they'll occupy the command line position where DOS expects to find *string2*, and will be passed to the wrong command.

The command lines within the batch file can contain other fixed parameters, as well. Say XCOPY was one of the commands the batch file was supposed to execute and you wanting it to XCOPY only new or modified files on a particular branch of your directory tree. If you'll recall, there's a switch restricting that command to new or modified files (/m), and, if that branch itself contains a subdirectory structure that you want to maintain in your copy, you might also want to use the /s switch. The line within your batch file, then, might look like this:

```
XCOPY %1 %3 /s/m
```

Here, with *string1* specified for the target drive and path and *string3* as the destination, you've simply tacked switches onto the command the same as if you had simply tacked the whole works in from the command line.

Also, variables can be passed through in a different order than the way they're entered on the command line. This isn't common, but there are instances when it's desirable—as in one case that'll be discussed later in this chapter, where in fact the ideal order actually passes the second string first and the first string later, as shown here:

```
command %2 %1 %3
```

The passing of variables with percent signs is one of the most powerful batch file tools DOS offers. It's one that every serious or even semi-serious DOS user ought to master—and it's easy.

%variable% (as used in batch files only)

Closely related but distinctly different and a little trickier is a different kind of variable you can use—a string set in the environment. Such strings can be accessed and used as conditional tests in batch files by simply enclosing the string in a pair of percent signs—that tells DOS this is no ordinary variable and to look for it in the environment. This can be particularly useful, as shown below:

IF %*environment_variable*% = = *some_string* (then do something)

This is another powerful batch file tool, one that I'll look at a little later, but for now I've opened up another can of worms—the double equal signs.

==

A batch file or macro can test to see if two strings match or don't match, and use the results of that test to determine which of two actions to take—for instance, performing a command or series of commands or aborting if the test results don't match what it's looking for. One equal sign won't do it, however. Just one of DOS's many little quirks.

@ (used in batch files only)

Since version 3.3, DOS has supported the use of the @ character to suppress the outputting of batch file command lines to the screen. Any line preceded by an @ will be suppressed. There has also been the option of using an ECHO OFF command in batch files to suppress the display of succeeding commands. Used at the top of a batch file, all you see on screen is ECHO OFF. By prefacing that command with an @, however (@ECHO OFF), the @ suppresses that line, the ECHO command suppresses the rest of the commands being displayed as run, and nothing reaches the display (with the exception of error messages generated by DOS).

$ (used in DOSKEY macros only)

Somewhat like the percent sign in batch files, the dollar sign takes on special meaning used in DOSKEY macros. Note, however, that for macro use, special codes must be substituted for traditional DOS symbols that are used in batch files (as shown in Table 9-1).

Nothing has no meaning

If you'll pardon what appears to be poor grammar, one of the things batch files can't handle is *nothing*. For instance, one of the handiest batch file tricks is to have DOS compare two values and, IF (or) IF NOT equal, to take some action. You can, for example, check against some variable you've set in the environment

Table 9-1 Special macro codes recognized by DOSKEY.

Code	Action
$g	Redirects output to file or device, equivalent to >.
$l	Redirects input from device or file, equivalent to <.
$b	Pipes output as input to other command, equivalent to |.
$t	Command separator. Use either when creating macros or entering commands directly.
$$	When dollar sign character is required; equivalent to $.
$1 - $9	Replaceable parameters, equivalent to batch file %1 - %9.
$*	Replaceable parameter, roughly equivalent to $1 - $9, except everything typed on command line after macro name will be substituted for the $* character.

as a test to determine whether or not to take some action, as in the following example that checks the data by trying to equate the value for TODAY—set as an environmental value during bootup—against a date value read as %4 from a file that was updated the last time the system was booted:

```
if %TODAY% = = %4 goto END
```

The problem is that, if for any reason no %4 parameter is found and DOS has nothing it can recognize to test against, DOS reads it this way:

```
if 03-15-1991 = = goto END
```

and just skips on to the end, defeating the whole purpose. So sometimes you have to give DOS a value to play with, even if it's a phony value that means nothing (but to DOS, at least, means something). It doesn't matter much what you use. Here, an exclamation point is something that would never appear in an actual value:

```
if !%TODAY% = = !%4 goto END
```

DOS reads this quite differently now and, this time, DOS gets to see and therefore do whatever commands it's supposed to do if the two values aren't equal:

```
if !03-15-1991 = = ! goto END
command
command
:END
```

This little cutie sees that my hard disk is optimized and MIRROR run the first time the system boots up for the day—but only once a day.

Great one-liners

To me, one of the slickest applications of batch files or macros is their ability to perform seemingly obtuse yet relatively simple tasks. Here's a typical one from my batch file directory—DIABLO.BAT. The name DIABLO is from the protocol

one of my printers uses. That printer is connected to OUT3, one of the output ports on a peripheral sharing box I use, so when I want to use that printer rather than my laser (which is the startup default, set from AUTOEXEC.BAT), all I have to do is type DIABLO at the DOS prompt. The batch file is as follows:

```
\DOS\UTILITY\ECHLPT /LPT1: @BFOUT3;
```

The @ character near the end of this command is part of the actual command, and isn't used to suppress the display of the command.

If you asked me to rewrite that one from memory, the best you'd get would be one of those "you've got to be kidding" looks. I looked up all that stuff maybe four years ago and wrote the batch file—and another one called LASER.BAT, and another one called DOT.BAT. So don't ask me to sit down and rewrite it. I'm not even sure I still know where the documentation is. But that's one of the beauties of batch files and macros.

This one, by the way, could be done just as easily with a DOSKEY macro, and the syntax to the right of the equal sign is identical to the batch file version:

```
DOSKEY DIABLO=\DOS\UTILITY\ECHLPT /LPT1: @BFOUT3;
```

In this case, the batch file happens to be the better choice because, unlike a DOSKEY macro, it can be executed from within my word processor without shelling to DOS. I think, probably, that the first time you try slipping one or more replaceable parameters into a batch file or a macro is the day you first appreciate the virtues of these tools.

A replaceable parameter is merely something DOS will recognize as someplace it's supposed to substitute another string or value when it reaches that point. In batch files, the most commonly used sign for variables is the percent sign (%) followed by a number 0–9. DOSKEY macros use the dollar sign ($) followed by a one-digit number, instead.

Exceptions to almost any rule

There's one peculiarity you can encounter when using replaceable values—one where batch files seem to violate a basic rule but DOSKEY macros don't. It has to do with text strings such as those you might input for FIND to search for. The syntax for FIND is:

```
FIND "string" filename[.ext]
```

with the string being any information enclosed by quotes that will fit on a single DOS line. One word or several, it makes no difference. It's the quote marks that matter. With that in mind, let's try to write a simple batch file for FIND using replaceable values—call it TEST.BAT:

```
FIND %1 %2
```

Easy enough: now let's put TEST to the test:

```
TEST "for that replaceable value" V:DOSPROGS
\dos\find "for that replaceable value" v:onceover.se5
---------- V:DOSPROGS
```

Think of SHIFT essentially as doing nothing you couldn't do by running a batch file that allowed only one replaceable parameter over and over in rapid succession, simply entering a new string—a different filename for example—for that replaceable value every time you run it.

Interesting. DOS has interpreted the four-word string enclosed in quotes as a single variable and passed it on that way to FIND. But when you try the same thing from a DOSKEY macro, you get a quite different result—this macro is called TEST, as well:

```
TEST "for that replaceable value" V:DOSPROGS
\dos\find "for that
FIND: Parameter format not correct
```

If you look closely, you can see what happened—notice the second quote mark is missing. The macro has taken *"for* as its first variable ($1) and replaced the second variable ($2) with that. Interpreting every blank space as the end of a variable (which DOS normally does), the macro has discarded the rest of the string before it ever even got to the part where you told it where you wanted it to look.

What you have here is a demonstration of something I told you at the beginning of the chapter—while there are many similarities and you can in most cases convert batch files directly into macros, there *are* differences.

The same name, but different

Frowned on by many "experts," one of my favorite batch file/macro tricks is giving batch files or macros the *same exact name* and *exact same syntax* as the naked command. Sound silly? Not really. Although it's usually an either/or situation, there are in fact several possible advantages to doing it this way.

Programs in directories not included in the path can be executed without having to include the path in the command. This keeps the PATH command short, and remember—a short path is a fast path. Also, by doing it this way, if the program happens to be in the default drive, DOS runs the program directly rather than going through the batch file; if the syntax is the same and parameters are passed in the same manner, it works the same either way.

Sometimes a command can actually be enhanced by going through a batch file. A classic case in point is the DOS FIND command, which will *not* accept wild card entries—the documentation tells you that in no uncertain terms. And

ordinarily you can't. But what about this one-liner I call FIND.BAT:

```
FOR %%A IN (%2) DO FIND %1 %%A /N/I %3
```

The %3 variable allows the output to be redirected to a file—especially useful to prevent scrolling off screen where multiple matches are anticipated. This is an especially interesting—and useful—batch file because, with the help of FOR, surely one of the most seldom used batch file commands, it can take one of DOS's traditionally more inadequate utilities, FIND.EXE, and make it downright useful by enabling it to search a whole directory, not just a single file.

The big *IF*

Just automating things so your computer can go through some particular exercise like a brain-dead zombie is one thing. Giving it enough "smarts" to know if something goes wrong—and act on it accordingly—is quite another. Writing *smart* batch files isn't really that much harder. And this is where batch files not only really go to work for you, but creating them is fun.

IF is the key to really unlocking batch file power. IF (*some_condition*) is met—if that *something* exists where it's supposed to—then something happens that won't happen if the condition isn't met. The test can take any of several forms:

```
IF [not] string1 = = string2 (command)
IF [not] EXIST (command)
IF [not] ERRORLEVEL (command)
```

The *command* can either be a DOS command (or another batch file) to be executed if the condition is met, *or* a direction (GOTO, for example) to find instructions for some other appropriate action.

GOTO is pretty self explanatory, but there's one quirk in the syntax that must be observed. GOTO points to a label and, although almost anything will suffice, a short descriptive word is usually the best choice, like:

```
IF [not] EXIST GOTO PROBLEM
```

Somewhere in the body of the batch file then—it doesn't really matter where—you need a routine that executes if the batch file jumps to the label :PROBLEM—and note that the label is prefixed with a colon here. LABEL used in this way must *always* be preceded by a colon—not at the GOTO end, but at the destination. So you have something like this:

```
:PROBLEM
command.....
optional additional command(s)
GOTO END
```

:PROBLEM causes some event or series of events to occur, and in this case jumps past anything and everything else that might be in the batch file to the label :END, typically the last line in a batch file. This is a beautiful escape mechanism.

IF EXIST . . .

. . . and NOT EXIST are probably the most popular conditional tools—and certainly about the easiest. How about:

```
IF EXIST B:*.* ECHO Disk in drive B: contains files
```

Put a blank disk in drive B: and you get no response. If there are any files you get the ECHO message. Admittedly not very useful in that form, but what if there's a particular disk with a particular file you want to be *sure* is there before going on? How about a little batch file like this:

```
:LOOP
IF EXIST B:filename.ext GOTO NEXT
ECHO Disk with filename.ext not found!
ECHO Put disk with filename.ext in drive B:
ECHO Press any key to continue
PAUSE
GOTO LOOP
:NEXT
next_command_or_subroutine
```

Now you've made it work for you. This won't let you continue unless the disk with the proper file is in the specified drive. Pressing any key will simply take you back to the top again (:LOOP), test again, and keep looping until you do put the proper disk in drive B:. Only then will it GOTO END.

Or you can use IF EXIST as a condition to determine branching with something like this:

```
IF EXIST B:file#1.ext GOTO ABC
IF EXIST B:file#2.ext GOTO CDE
GOTO END
:ABC
COMMAND
GOTO END
:CDE
COMMAND
:END
```

Here you've set the stage for any one of three things to happen:

- If *file#1.ext* exists, you'll jump directly to label :ABC and the next two lines won't even be read.

- If *file#2.ext* is found, control will pass to a branch labeled :CDE.
- If neither condition is met, the batch file will jump directly to label :END, skipping everything in between.

IF [NOT] ERRORLEVEL

ERRORLEVEL is one of DOS's handiest little helpers, and yet it's surprising how few users understand it, let alone really take advantage of it. It lends itself especially to use in batch files—which is, in fact, about the only good use the average DOS user can put it to. But that's enough.

Suppose you want to write a batch file that will back up certain critical files to a floppy disk at the beginning of every work session—every time you boot your system—and make sure you have those copies before proceeding. You could create a batch file along the lines of the one here that you could call from your AUTOEXEC.BAT:

```
@ECHO OFF
:LOOP
IF EXIST d:filename GOTO BACKUP
ECHO Backup disk not found in proper drive
PAUSE
GOTO LOOP
:BACKUP
XCOPY d:\directory\*.* /S d:
IF ERRORLEVEL 5 ECHO Aborted: probable error reading or writing disk
PAUSE
GOTO...
...
...
IF ERRORLEVEL 1 ECHO No files found to copy
GOTO END
IF ERRORLEVEL 0 ECHO Backup completed without error
:END
```

Here I've combined several tricks into the same batch (or subroutine, depending on how you choose to use it). I've made the program keep looping back and retesting the target disk until the proper disk is in place before executing XCOPY. But then I used ERRORLEVEL to be sure that the copy process was completed satisfactorily with different ERRORLEVELs triggering different responses—or, if successful, that it simply echoed an appropriate message and terminated.

Not all DOS commands return ERRORLEVELs when they terminate, but most of the ones where it's really of benefit do, it seems. It's easy to test to see if an ERRORLEVEL value has been returned with a simple batch file like the one

```
@ECHO OFF
IF ERRORLEVEL 10 ECHO ERRORLEVEL is 10
IF ERRORLEVEL 10 GOTO END
IF ERRORLEVEL 9 ECHO ERRORLEVEL is 9
IF ERRORLEVEL 9 GOTO END
IF ERRORLEVEL 8 ECHO ERRORLEVEL is 8
IF ERRORLEVEL 8 GOTO END
IF ERRORLEVEL 7 ECHO ERRORLEVEL is 7
IF ERRORLEVEL 7 GOTO END
IF ERRORLEVEL 6 ECHO ERRORLEVEL is 6
IF ERRORLEVEL 6 GOTO END
IF ERRORLEVEL 5 ECHO ERRORLEVEL is 5
IF ERRORLEVEL 5 GOTO END
IF ERRORLEVEL 4 ECHO ERRORLEVEL is 4
IF ERRORLEVEL 4 GOTO END
IF ERRORLEVEL 3 ECHO ERRORLEVEL is 3
IF ERRORLEVEL 3 GOTO END
IF ERRORLEVEL 2 ECHO ERRORLEVEL is 2
IF ERRORLEVEL 2 GOTO END
IF ERRORLEVEL 1 ECHO ERRORLEVEL is 1
IF ERRORLEVEL 1 GOTO END
IF ERRORLEVEL 0 ECHO ERRORLEVEL is 0
IF ERRORLEVEL 0 GOTO END
:END
```

9-1 Batch file for determining errorlevel returned by programs that support errorlevel detection. This must be run after program has been terminated in various ways, and tests repeated to assure results when not documented.

listed in Fig. 9-2, which tests until it finds a match, returning an appropriate message.

Note, however, that the program starts testing from the highest number down. You must, when using multiple ERRORLEVEL conditionals for branching, always put the highest possible valid error level you want to test for first and work down.

Macro or batch file?

While I find that it sometimes takes a little experimentation to translate good batch files into workable DOSKEY macros, in the long term the choice often comes down to something like picking petals off a daisy. There are exceptions, of course.

As a general rule of thumb, if a task requires more than two or three lines to perform from a batch file, it probably isn't a good candidate for macro execution. This in itself limits the applicability of macros, particularly where conditionals are involved. This isn't to say that you can't use conditionals in macros, but you're

clearly more limited to the direct execution of a single command—or aborting the process—on the basis of a simple test, as in:

```
DOSKEY TEST1 = if exist e:test CHKDSK C:
```

There's more involved than just the line-length limit, however. The manner in which macros must execute multiple commands from the same command line is just different from the way a batch file executes, as in this example:

```
DOSKEY TEST2 = if exist e:test goto end $techo Test must not exist
$t:end
```

A batch file with each command on a separate line would have no difficulty with this one, but the macro (including DOS prompts) would run like this:

```
C:\
test2
C:\
if exist e:test goto end
C:\
echo Test must not exist
Test must not exist
C:\
:end
Bad command or file name
C:\
```

Admittedly, a batch file written on a single line with the commands separated by the piping character (|) has the same problem as the single-line macro, but runs exactly as expected when spread over three lines. With a batch file you have that option.

Also, batch files (even short ones) aren't necessarily directly interchangeable with equivalent DOSKEY macros, however. As mentioned earlier, one of my favorite one-liners, FIND.BAT, that does what Microsoft says FIND can't do, isn't interchangeable I found. Below I've offset FIND.BAT so you can easily compare it, character by character, with the DOSKEY macro on the line below it.

```
                    FOR %%A IN (%2) DO FIND %1 %%A /N/I %3
DOSKEY FIND = FOR %%A IN ($2)  DO FIND $1  %%A /N/I $3
```

Note that, in both cases the third string from the command line on the line is inserted ahead of the second string. This is done to keep the syntax identical to what it would be if you were using FIND.EXE directly. Not a trick you need fall back on very often, but a handy one to keep in mind.

In any case, in this form the macro will work and, to a point, does exactly

what the batch file does. But only to a point. The FOR part works fine, allowing you to input a whole subdirectory instead of a single filename as a variable (%2) for FIND—*except* when the string I tell it to look for contains more than a single word. The syntax for FIND, if you'll recall, is:

FIND *"string"* [*d:*][*path*][*filename*[*.ext*]]

The "*string*" input variable for FIND normally allows the input of more than just a single word—any string that, combined with the rest of the command, doesn't exceed the length of a DOS line—provided the string is enclosed in quotes. DOSKEY parses commands a little differently however, and as a macro allows only a single word within the quotes, treating any additional words as separate variables.

Also, if I want to redirect the output to a file with the macro version, it's necessary to use two redirection character codes, > > (to append), rather than the single > (simple redirection) that works in the batch file equivalent. For some reason, a single redirection arrow used in the macro causes the output of each file searched to overwrite the output of the previous one. The $g doesn't seem to work here and, if no redirection is wanted, it cause the macro to abort, issuing a file creation error message.

There are clearly significant differences, and while these might not be apparent to the casual batch file users moving over into doing macros, you'll see these differences when using more advanced techniques.

Also, remember that not everything that's legal and valid in batch files is acceptable when writing DOSKEY macros. A couple of notable exceptions are the ECHO OFF command and also the @ character to suppress outputting specific commands to the screen.

ECHO OFF is just totally ignored by macros and the @ ahead of any command causes DOS to return a *Bad command or filename* message. ECHO can otherwise be used to echo messages just like from a batch file—but with other information displayed as well, as in the example shown below. The macro was created with the line:

```
DOSKEY TEST = \DOS\XCOPY $1 $2 $3 $T IF ERRORLEVEL 0 ECHO
WHOOPEE
```

and, when run, it looks like this on screen:

```
TEST V:DISK__MAN.CH E:
\DOS\XCOPY V:DISK__MAN.CH E:
READING SOURCE FILE(S)...
V:DISK__MAN.CH
      1 File(s) copied
IF ERRORLEVEL 0 ECHO WHOOPEE
WHOOPEE
```

Whoopee nothing. I'd call it just pretty messy—and confusing. By comparison, a similar batch file with @ECHO OFF as its first line runs nice and clean:

```
TEST V:DISK_MAN.CH E:
Reading source file(s)...
V:DISK_MAN.CH
      1 File(s) copied
WHOOPEE
```

Here, both do exactly the same job and one is faster, the other clean. You pays your money, you takes your chances . . .

Other considerations

While disk space isn't generally as critical with hard disks as with floppies, even today's bigger hard disks tend to fill up rather quickly and DOSKEY macros have it all over batch files when it comes to storing them. The smallest batch file—a single word, a single letter even—takes 2048 bytes of hard disk space to store. That's one allocation unit; the smallest space DOS ever uses on a hard disk.

Typically, in my batch file directory I have 47 files, most of them short, and the total disk space actually used by *all* of them amounts to only 7907 bytes—not really very much space. If I converted all of them to macros, they could all fit nicely into a single file of no more than five allocation units, or 10240 bytes (don't forget the 47 times you'd have to start lines DOSKEY *macroname*=). As 47 separate batch files, however, they take up 192,512 bytes—over 192K of hard disk space— or roughly as much as 200 pages of manuscript. That means that, saved as separate batch files, you have about 95% of wasted space.

Sometimes you have to look beyond the obvious to put things in their proper perspective. Still, as you've seen, there remain many situations where macros just won't cut it and the only way to do the job is good old fashioned batch file programming. And now that you know how, perhaps the biggest trick is knowing when . . . unless you've got a daisy handy.

CHAPTER 10

Memory management above and beyond

DOS 5.0 is more than just a new release of DOS. It's a statement on the future of desktop processing using DOS machines. DOS 4.0 offered token support for extended and expanded memory, but of too limited a nature to be taken seriously. This time DOS is serious and the future as DOS sees it is in extended memory—linear memory that starts at 1024K (1 megabyte) where DOS itself must stop, and continues in an unbroken string to addresses and numbers that boggle the imagination. No longer are we just talking millions of bytes of RAM, but rather billions.

Of course the very mention of extended memory excludes the large installed base of 8088 machines—machines that for years have been able to break the 640K barrier by reaching out into expanded memory, which is similar to extended memory only in the fact that both use RAM. Philosophically and practically they're completely different.

So why, do you ask, should DOS then jump on the extended memory bandwagon to the exclusion of such a large installed user base when there's a mechanism in place by which *all* DOS machines can share a common memory platform? The answer to that one is too complex for more than a superficial explanation here, but in essence it comes down to severe limitations that are inherent in expanded memory.

Expanded memory has already gone about as far as it can go. Even with expanded memory, you are and always will be limited to programs that can run

within the 640K you're accustomed to with DOS. You can play games with it, swapping chunks of running code and data in and out and, with those games, you can run programs you couldn't run otherwise, but only to a point.

Another serious limitation lies in the fact that the 80386 and now the i486 are 32-bit chips that can never achieve their full 32-bit processing power potential from within DOS—which is where all expanded memory processing is conducted.

The emergence of a new kind of software called the *DOS extender* changed all that, however, by allowing programs to load from DOS but then blast off into protected mode in extended memory to do their actual processing out there where, depending on the design of the CPU chip and DOS extender being used, run wide open and, on 386 and higher chips, actually unleash the full 32-bit processing power on programs that can be many megabytes in size. So extended memory is only just beginning to fulfill its promise.

The 80286 also has extended memory capabilities, and it was in fact initially heralded as the dawn of the new era. That promise was dealt a severe setback as a result of design problems with the chip itself, however, and it's been only fairly recently—since the emergence of extended memory teamed with the raw processing power of the 386 (and higher) chips—that many developers have taken a new look at the 80286. And even though some nagging problems still remain, DOS extenders written especially for the peculiarities of the 286 have found ways of getting around most of the worst of the 286 problems.

As for the 8088, it still is—and always will be—excluded from the club. It's limited by design to a one-megabyte world. This is why the core of DOS itself is a strictly one-megabyte operating system that in this release, like all previous releases and all DOS versions yet to come, will continue to run on 8088 machines, as well as on the hottest new chip of the day. DOS and the 8088 are inseparable.

And clearly, with new 8088 machines still being manufactured, adding to the tremendous user base already in place, expanded memory isn't going to go away. In fact, there are some functions associated with expanded memory that can't be duplicated by extended memory—important functions really only applicable to the 80386 and higher chips.

What DOS 5.0 has done, then, is add an optional extra level—actually two extra levels—of support for memory beyond the normal limits of DOS:

- HIMEM.SYS, an extended memory manager that can be used on all 80286, 80386, and higher machines.
- EMS386.EXE, an LIM 4.0 EMS *emulator* for use only on 80386 and higher systems that can make some assigned chunk of extended memory act like expanded memory.

These two, then, support two separate but related memory functions unique to the

basic memory form they represent—extended memory *per se* and the High Memory Area (1024K−1088K) as functions of HIMEM.SYS and ordinary expanded memory, and memory that can be mapped to unused DOS address space between 640K and 1024K as functions of EMM386.EXE (but, again, only on an 80386 or I486 machine).

Grass roots support

HIMEM.SYS is the key to the entire DOS 5.0 memory management scheme. While it paves the way to extended memory on 80286 and higher machines, one of its most valuable contributions to DOS is not the extended memory access in itself, but rather access to the High Memory Area (HMA). This is that mysterious extra 64K segment of DOS memory someone discovered several years ago. (Quarterdeck Office Systems, developers of DESQview, were apparently the first to discover that this 64K area could be accessed in real mode from DOS, although it lies almost entirely beyond DOS's traditional one-megabyte limit.)

I'm putting this one first because you don't necessarily have to use DOS's HIMEM.SYS to be able to load DOS to the HMA. The ability of DOS 5.0 to load much of the kernel to the HMA is intrinsic to the DOS kernel itself (through IO.SYS) and you can use the DOS=HIGH command to load the kernel to the HMA in conjunction with any 3rd-party memory manager that supports the High Memory Area.

From 1024K, where DOS normally leaves off, to within a few bytes of 1088K is an area that can be legitimately accessed by DOS in real mode despite the fact that the memory involved lies in what would ordinarily be extended memory, and beyond the reach of DOS. Because this memory isn't contiguous to conventional DOS memory, separated from it by the video, ROM, and other system functions, it can't simply be added to the nominal 640K to give you an additional 64K of workspace at the top.

What DOS 5.0 has done is to rewrite the DOS kernel in such a way that DOS can relocate 47K of its own bulk up into the High Memory Area (HMA), giving you an extra 47K down near the bottom of conventional memory to start loading from. The top is still 640K, but the total free memory for your applications to use is some 47K greater, as shown in Fig. 10-1.

With DOS 5.0 installed in conventional memory, you have a little over 584K available. But look what happens if you include the following two lines to the CONFIG.SYS of a machine that supports extended memory, and has the memory installed to work with—here I'm assuming the use of HIMEM.SYS rather than any other memory manager:

```
DEVICE=[d:][path]HIMEM.SYS
DOS=HIGH
```

With these lines, you've both installed HIMEM.SYS to access installed memory

beyond 640K as extended memory (and support the High Memory Area), and told DOS to try to load itself into the High Memory Area. As you can see in Fig. 10-2, by kicking most of DOS upstairs, you've increased the amount of contiguous DOS memory available to load and run applications over 635K.

Admittedly, in both cases you're dealing with a stripped-down system and, as soon as you start loading even some DOS TSRs like MODE, you'll see the numbers start to shrink. The point, though, is that in this back-to-back comparison on the same machine you've got an extra 51K to start with. This is a very major increase.

To have a High Memory Area, you must first have extended memory, which is where HIMEM.SYS enters the equation (and where we leave the 8088 behind). It's also where you have to start playing some games.

Burning the candle at both ends

Extended memory, as I've defined it, is an unbroken string of contiguous memory starting at 1024K and continuing, essentially, as far as your craving for power—and your bank account—allows. Extended memory can be accessed, however, in two distinctly different ways—either from the top or from the bottom. For most purposes, it's preferable to work from the top down.

Any kind of memory that can be used by DOS—and the watchword here is *by*, not in *conjunction with*—has to be contiguous to DOS, and the bottom of extended memory is. So for use by DOS, you have to take the memory that's right next to it at the bottom the extended memory stack. Extended memory from the top, the HMA from the bottom—sort of like burning a candle at both ends.

There's more to it than that, of course. By way of explanation without going into a lengthy discourse, DOS, for reasons that go back before there was a DOS (and that are no longer valid), works in terms of 64K memory segments. A 64K memory segment, however, doesn't have to start at an address that's a multiple of 64K—or a multiple of much of anything else for that matter. So what the HMA is, in simple terms, is a 64K memory segment that starts just barely within DOS's one-megabyte nominal limit.

As long as the segment has a starting address within DOS's grasp, DOS can use the whole segment. The starting address used by the HMA is only 16 bytes below 1024K, so 64K starting from that point takes up to just 16 bytes shy of 1088K. Of course, 8088s simply don't have address pins allowing them to work beyond one megabyte, so that's where they come up short.

The 80286 (and 80386SX) can deal with linear addresses to 16 megabytes, and the 80386DX and i486 all the way to 4 gigabytes, which is plenty of room to allow for dealing 64K off the bottom and still have plenty of address space to spare to support more extended memory megabytes than most users will ever need or likely ever want to pay for.

First come first served

Unlike any of the other memory areas available to DOS users, only one program can use the High Memory Area. This is becoming increasingly important to understand as the number of programs looking to the HMA for extra space continues to increase. It's also increasingly important to understand just what's at stake when granting access to the HMA—and it *is* a granted right, not one that any program—even DOS—has automatic access to.

So you have one 64K memory segment that can support only one program. Given that, you might have to make a conscious choice between which of several contenders it should go to. DOS, of course, would like to have it, and will, in many cases, add both the DEVICE=HIMEM.SYS and DOS=HIGH statements to your CONFIG.SYS during the DOS 5.0 installation process. And for most users, I'd say let DOS have it.

There are exceptions, however, Quarterdeck's DESQview being the most classic example. While I demonstrated the 47K gain promised by loading most of the DOS 5.0 kernel to the HMA, DESQview, working in conjunction with Quarterdeck's QEMM memory manager, can in fact use no less than 63K of the available 64K. A DESQview user, allowing DOS to have the HMA rather than DESQview, would then actually lose in the exchange and have *less* conventional memory left to run application software. In fact, in practice the actual loss is greater than the numbers would seem to indicate. To my knowledge, no one else comes even close to utilizing the HMA that fully, but it makes an important point.

Realistically then, unless you have no other software that can use the HMA, you should run some tests and see which combination will give you the greatest amount of free conventional memory to run your applications in. All it takes is a little trial and error and one of the newer DOS utilities called MEM (MEM.EXE) to check it out. I used MEM to obtain the comparative data in Figs. 10-1 and 10-2, and while it supports the use of several switches to access more advance features, typing just plain MEM and Enter at the keyboard is all you need for this task.

```
655360 bytes total conventional memory
655360 bytes available to MS-DOS
584576 largest executable program size

3407872 bytes total contiguous extended memory
3407872 bytes available contiguous extended memory
```

10-1 MEM report on available memory with DOS 5.0 loaded in conventional memory.

When testing, remember that DOS now supports the use of REM in the CONFIG.SYS—a very handy way to make temporary changes without com-

```
  655360 bytes total conventional memory
  655360 bytes available to MS-DOS
  635648 largest executable program size

3407872 bytes total contiguous extended memory
      0 bytes available contiguous extended memory
3342336 bytes available XMS memory
        MS-DOS resident in High Memory Area
```

10-2 MEM report on same system shown above but with approximately 47K of DOS kernel relocated to HMA (1024K – 1088K).

pletely screwing up your configuration, as shown here:

```
DEVICE = C: \ DOS \ SETVER.EXE
REM DEVICE = C: \ DOS \ QEMM \ QEMM386.SYS .....
REM DEVICE = C: \ DOS \ ......
.
.
.
.
SHELL = C: \ DOS \ COMMAND.COM C: \ DOS \  /p
DEVICE = C: \ DOS \ HIMEM.SYS
DOS = HIGH
```

Even better yet, make a bootable floppy with its own CONFIG.SYS and work from it until you're satisfied, only then making any changes you decide on to your hard disk CONFIG.SYS.

As you can see, the HMA is a unique resource and one that, if used correctly, can serve you very well.

HIMEM.SYS, the easy part

By far, the easiest memory to manage is plain old ordinary extended memory. There's really nothing to it and, as extended memory managers go, the new DOS 5.0 HIMEM.SYS is as good as any—no better or worse than the competition. What's most interesting about HIMEM.SYS is that it reflects Microsoft's philosophical commitment to extended rather than expanded memory for the future of DOS.

By contrast, some of the better-known 3rd-party memory managers, dating conceptually to a time before extended memory was even a viable option, convert any memory available to them into *expanded* rather than extended memory. These—including at least one that does support the HMA (most don't without some outside help)—relinquish some of the memory available for use as extended memory only by special request.

Not all extended memory managers support the High Memory Area discussed in the previous section. Even though it uses a piece of extended memory, the HMA does require specific management features not incorporated in all extended memory managers.

Anyone who bought a 286 or higher machine with one meg (or more) of memory installed as it came from the factory who doesn't currently have a memory manager of any sort installed should pay special attention here. If you bought a meg or more of RAM—and most machines these days come with at least one meg—and if you don't have a memory manager, you're using only 640K and nobody, including your software, even knows the rest of it is there.

Unfortunately, a lot of users fail to understand this, and as a result there are a lot of 640K machines out there with gobs of RAM being wasted. Add an extended memory manager to a 640K machine with one full meg installed and suddenly you have 384K of expanded memory you didn't have before.

In today's big numbers world, 384K sounds pretty small, considering that an 80386 can handle no less than 4,294,967,296 bytes of RAM. But the HMA can use only 64K, leaving you at least 320K to build at least a modest RAM disk (RAMDRIVE.SYS) and a disk cache (SMARTDRV.SYS), which is enough to make a difference in performance.

Here, for example, are typical CONFIG.SYS entries that would take a 286 (or higher) machine with one meg of installed RAM and turn an unused 384K into extended memory with HMA support, load most of the DOS kernel to the HMA, and create a 256K RAM disk and 64K cache:

```
DEVICE = C: \ DOS \ HIMEM.SYS
DOS = HIGH,UMB
DEVICE = C: \ DOS \ RAMDRIVE.SYS 256 /e
DEVICE = C: \ DOS \ SMARTDRV.SYS 64
```

And with DOS 5.0, it doesn't even cost you any extra. Compared to the going price for 3rd-party memory managers that only manage memory, HIMEM.SYS alone could almost justify the cost of the upgrade kit.

By the way, users of Windows 3.0 might recognize the name HIMEM.SYS as also being one of the files associated with this release of Windows (generally residing in the WIN subdirectory). While in fact similar in a number of respects, the DOS 5.0 version of HIMEM.SYS is an enhanced version, and Windows users currently using the HIMEM.SYS that came with Windows should replace it with the more powerful DOS 5.0 version.

Only for the upper class

In addition to offering an XMS extended memory manager for the first time in this release, MS-DOS 5.0 for the first time offers expanded memory emulation, as

well—full LIM 4.0 EMS expanded memory emulation, but only for use on 80386 and higher machines. MS-DOS has provided no equivalent to the XMA2EMS .SYS and XMAEM.SYS drivers that made their appearance in 4.x PC-DOS. This despite the fact that even 8088 PCs and clones can use expanded memory.

Most of those 8088 and even many 80286 machines, however, can have memory available for use beyond 640K only by adding expansion cards such as the AST RAMpage, and these generally provide their own proprietary memory management software. Viewed in this context, then, the limiting of expanded memory support to the upper level of machines and catering specifically to features found only on those machines makes sense.

Here again you see the philosophy of Microsoft's approach to memory management—an approach that puts extended memory first, rather than saying "oh yes, and by the way . . ."

To have expanded memory using DOS 5.0's new scheme, you first must have extended memory. This isn't an option. The new expanded memory emulator, EMM386.EXE, has to find a chunk of extended memory it can manage in a way that makes you, your computer, and your software *think* you have expanded memory. You really don't, but it doesn't matter as long as everybody thinks you do.

So because EMM386.EXE will function only when it has extended memory to work with, it's critical that HIMEM.SYS be installed ahead of it, in this manner:

```
DEVICE = HIMEM.SYS
DEVICE = EMM386.SYS
```

This isn't to say that EMM386.SYS absolutely has to be the very next entry after HIMEM.SYS. It doesn't. You can slip RAMDRIVE.SYS or SMARTDRIVE .SYS, or both, or anything else in there ahead of EMM386.SYS—anything, that is, as long as HIMEM.SYS is loaded in *ahead* of EMM386.EXE or anything else that uses extended memory. Again, you have to have extended memory before you can use it.

And by the same token, any devices that are going to use expanded memory *must* be loaded after EMM386.EXE because, until it's in there, you don't have expanded memory. There's a definite pecking order here that absolutely must be observed.

Grabbing unused memory

The 80386 brought with it more than just the power to use extended memory *as* extended memory. It brought with it the capability of having LIM 4.0 EMS memory mapped (assigned) to any DOS-addressable address space. This opened up a whole new area to DOS above 640K, sometimes as much as another 100K or more just in this area alone that DOS can use for TSRs, device drivers, etc., (and

therefore free the space they would otherwise occupy down in conventional memory for use by applications).

Despite the fact that everything above 640K has traditionally been set aside for system use, there's a lot of wasted space up there—a little bit here, a little more there (being careful, of course, not to trample on the video, the ROM, or any other system functions). Actually, there's usually a fairly significant area in the ROM region (F000h–FFFFh) that can be mapped over and used. It gets a little tricky finding it, however, and EMM386.EXE doesn't try. There are usually two or three pretty decent chunks, though, and if you can find them and map LIM 4.0 EMS memory to the unused address spaces in those areas, there's room to load some TSRs and device drivers up there instead of in conventional memory.

To map expanded memory to these unused addresses, you need to add a statement to your CONFIG.SYS. You can either make separate statements, like this:

```
DOS=HIGH
DOS=UMB
```

or combine them in a single statement, like this:

```
DOS=HIGH,UMB
```

You normally wouldn't use the UMB statement in conjunction with 3rd-party memory managers. While it might work, other managers usually have their own command syntax, and the use of this command could cause a conflict. Earlier in this chapter, I showed as much as 635K of space available to load application software and their data on a bare-bones system. Now I'm going to put a little meat on those bones and *still* have nearly that much usable space to work with.

The key to loading device drivers into upper memory blocks—as memory mapped to high DOS address space is called—is the new DEVICEHIGH function in IO.SYS (where the DEVICE command also resides). Again, because this function is intrinsic to DOS itself rather than EMM386.EXE, it isn't exclusive to EMM386.EXE, but can be used with other memory managers as long as those memory managers provide the LIM 4.0 EMS support required. On a 386 with proper 4.0 EMS support, you simply use a DEVICEHIGH= command in your CONFIG.SYS, instead of a DEVICE=.

And it really is as simple as that. At least until you try packing ten pounds in a five pound bag. Remember, you're just taking bits and pieces of left-over unused address space up there. How much is there? The question really isn't how much is there, but rather how much is there that's usable. EMM386.EXE isn't the most powerful tool available to work with, but will typically come up with something like what's shown in Fig. 10-3.

You've really got to read fast (or freeze the scrolling screen with Ctrl–NumLock) to read this as the system is booting. That whiz-by opening screen really isn't the information you need, anyway. You're better of using MEM again,

```
EMM386 successfully installed

  Available expanded memory . . . . . 512 KB

  LIM/EMS version . . . . . 4.0
  Total expanded memory pages . . . . . 28
  Available expanded memory pages . . . . . 4
  Total handles . . . . . 64
  Active handles  . . . . . 1
  Page frame segment  . . . . . E000 H

  Total upper memory available  . . . . . 131 KB
  Largest Upper Memory Block available  . .  75 KB
  Upper memory starting address . . . . . B000 H

EMM386 Active
```

10-3 Opening screen showing status of EMS memory on typical 80386, in this case with 131K mapped to unused address spaces above 640K and available for loading device drivers and TSRs.

only this time with the /c switch (for classify). You might as well try and load some drivers up there while you're at it, however, so you have something more to look at. Here is an excerpt from an actual CONFIG.SYS. Let's see how well EMM386.EXE does with it:

```
DEVICE = HIMEM.SYS
DEVICE = EMM386.EXE 2048 RAM
DOS = HIGH,UMB
DEVICEHIGH = RAMDRIVE.SYS 720 /E
DEVICEHIGH = SMARTDRV.SYS 360
DEVICEHIGH = EGA.SYS
DEVICEHIGH = STAK\STACKER.COM /B = C800 /M = 0 D:\STACVOL.000
```

These are the commands. And Fig. 10-4 is what you'll get when you specify MEM /c at the C>.

What this shows is that you've loaded all four of those device drivers plus two small TSRs (from the AUTOEXEC.BAT) into upper memory blocks, and you *still* have something over 30K to use up there. Not bad for an afternoon's work, and even better considering it took only minutes—just long enough to add the EMM386.EXE driver and the UMB, change DEVICE= to DEVICEHIGH=, and reboot.

This is a fairly easy case, but by the same token fairly typical and clearly well within the capabilities of anyone who can write and edit a CONFIG.SYS. Even without getting fancy, there should be enough power here to make most users happy—probably for some time to come.

```
Conventional Memory :

  Name                 Size in Decimal          Size in Hex
------------      ----------------------   -------------
  MSDOS                 15536      ( 15.2K)       3CB0
  HIMEM                  1184      (  1.2K)        4A0
  EMM386                 8400      (  8.2K)       20D0
  COMMAND                2624      (  2.6K)        A40
  FREE                     64      (  0.1K)         40
  FREE                 627344      (612.6K)      99290

Total  FREE :          627408      (612.7K)

Upper Memory :

  Name                 Size in Decimal          Size in Hex
------------      ----------------------   -------------
  SYSTEM               184320      (180.0K)      2D000
  RAMDRIVE               1184      (  1.2K)        4A0
  SMARTDRV              17904      ( 17.5K)       45F0
  EGA                    3280      (  3.2K)        CD0
  STACKER               22336      ( 21.8K)       5740
  MODE                    464      (  0.5K)        1D0
  XYKBD                  1552      (  1.5K)        610
  FREE                     48      (  0.0K)         30
  FREE                  30880      ( 30.2K)       78A0

Total  FREE :          30928      ( 30.2K)

Total bytes available to programs (Conventional+Upper) :     658336   (642.9K)
Largest executable program size :                            627216   (612.5K)
Largest available upper memory block :                        30880   ( 30.2K)


 458752 bytes total EMS memory
  65536 bytes free EMS memory

3407872 bytes total contiguous extended memory
      0 bytes available contiguous extended memory
1986560 bytes available XMS memory
        MS-DOS resident in High Memory Area
```

10-4 Report of memory status using MEM with /c switch. With four device drivers and two TSRs loaded in UMBs, over 30K is still available in upper memory on an 80386.

Still, they won't all go this easily. There are some nasty programs out there that require a lot more memory during loading than they have to need to actually run. To make things fit, sometimes you have to juggle things around. Sometimes you have to change the loading order by changing the order things are called for in the CONFIG.SYS (or AUTOEXEC.BAT for TSRs). And sometimes, when all else fails, you simply have to let the overflow load low—which then comes down to trying to squeeze the most bytes into upper memory that you can, so you have the fewest spilling over.

At first glance, it appears that some of the top 3rd-party managers are more efficient at locating unused address space above 640K and mapping to it—and they're clearly more aggressive in their approach. EMM386.EXE, for example,

by default doesn't map to any address space below C800h or above E000h, leaving recoverable areas unused on most machines.

EMM386EXE does support an option to include any other address spaces that you know exist up there. For instance, on one of my machines there's 32K open between B000h and BFFFh, which could be picked up by adding it to the EMM386.EXE line in the CONFIG.SYS this way:

DEVICE = C:\DOS\EMM386.EXE 20 RAM I = B000-BFFF X = C800-CBFF
FRAME = E000

There is, on that machine, at least another 24K that can be mapped over in the ROM region without upsetting the system—in fact, it's included in the 131K total shown on an earlier screen. And that and any other known gaps can be covered by adding one or more additional I = *nnnn-nnnn* statements, making EMM386.EXE at least potentially as powerful as any EMS manager on the market. That's the good news. Unfortunately, DOS doesn't provide a decent utility for finding these holes to work with. At this point, you need something like Quarterdeck's Manifest for that.

And of course when you get into having multiple blocks of mapped memory to work with, working out a loading sequence that packs the most in becomes more of a problem, too. MEM can help with that one, though, through another switch it supports, /d (for /debug), which reports the starting addresses (left-hand column) for every item loaded, as shown in the excerpt in Fig. 10-5.

So even though DOS 5.0 lacks some of the amenities to make life easier above 640K, the power is there. In fact, I was pleasantly surprised how well the HIMEM.SYS/EMM386.SYS duo really does—or can do, anyway. With just a little help from you.

```
01E4D0          MSDOS           081B10      -- Free --
09FFF0          SYSTEM          02D010      System Program

0CD010          IO              00AEE0      System Data
                  RAMDRIVE      0004A0       DEVICE=
                    E:                        Installed Device Driver
                  SMARTDRV      0045F0       DEVICE=
                    SMARTAAR                  Installed Device Driver
                  EGA           000CD0       DEVICE=
                    EGA$                      Installed Device Driver
                  STACKER       005740       DEVICE=
                    F:                        Installed Device Driver
0D7F00          MSDOS           000030      -- Free --
0D7F40          MODE            0001D0      Program
0D8120          XYKBD           000040      Environment
0D8170          XYKBD           0005D0      Program
0D8750          MSDOS           0078A0      -- Free --
```

10-5 Excerpt from MEM report using /d (debug) switch. This identifies specific addresses in use, essential in many cases to develop loading strategy to maximize use of UMBs on 386 and higher systems.

CHAPTER 11

Excess baggage

It's amazing the amount of excess baggage many users have cluttering their hard disks—DOS baggage. Much of that, of course, comes from a general lack of understanding of what DOS really is and what it isn't—which is why I've belabored that point. It's only with that knowledge that you can trim DOS down to fighting trim, which is more important in DOS 5.0 with its 2+ megabyte default installation than ever before.

2,036,532 bytes—that's how much 5.0 I had on my hard disk after installation—and that didn't even include COMMAND.COM or the hidden files—the *real* DOS. I'm down at this point to only 921,175; I've saved 1,115,357 of hard disk space, and yet I haven't given up a single DOS command or function that I ever use. In fact, if anything I've been less than frugal, keeping several items that I rarely use.

What's alarming is the way DOS has crept up in size on us, making an issue of something that was never seriously an issue until now. With 3.3, the total size of DOS on my machine was 147,983 bytes, scaled down from something just a little over 600,000 as initially installed. So even after clearing out a lot of excess baggage, I *still* have 7 1/2 times as much 5.0 as I had 3.3. These are serious numbers, so it's time to get down to some serious housecleaning.

If you're squeamish about dumping parts of DOS for fear Bill Gates will jump right out and grab you in your sleep some night, there's a cautious way to go about it. One of my favorite—and one of the most-used—directories on my hard disk is C:\DOS\JUNKYARD. I don't know how I ever got along without it. When in doubt if I'll ever need a file or program, it's off to the JUNKYARD. It's still on my disk in case I really need it, but JUNKYARD isn't on my path.

This JUNKYARD subdirectory is sort of an out-of-sight, out-of-mind place, and things stay there for quite a while before I finally dump them. Usually, in fact, until I find I'm running short of hard disk space again—a regular occurrence

around here, even with bigger and bigger hard disks. At that point, I go to the JUNKYARD and start throwing stuff. By definition, anything that's still there is something that I haven't used, sometimes for quite a while—sometimes I can't even remember what a program was, so those go out for sure.

So if you want to play it safe, set up a JUNKYARD; call it anything you like. And then start trimming things off of DOS. Of course, there's nothing you can do to rid yourself of any of the commands or functions DOS supports internally, certainly none of those that IO.SYS supports directly, and those supported by COMMAND.COM can only be discarded by replacing COMMAND.COM with some alternate interpreter, which opens up a whole new can of worms. I'll let that one slide for now, however, and stick with just the DOS externals.

And remember, too, that if you should erase a file that you later find you need—or if you find that your needs have changed—you can always go back to the distribution disks and use EXPAND.EXE to uncompress the packed file that you need. The packed files have the same names, except that the last character of the extension is an underline character, as in .EX_ or .CO_. So that's all the more reason for you to whittle the DOS installed on your hard disk down to size.

So let's start off by figuring out just what you *do* need to keep handy. The list in Fig. 11-1 will vary according to your hardware platform (8088, 286, or 386 and up), level of experience, and individual working habits, but here's what's still on my hard disk. It might not suit your needs exactly, but it's a place to start.

```
ATTRIB   .EXE      FIND     .EXE      QBASIC   .HLP
BACKUP   .EXE      FORMAT   .COM      RAMDRIVE.SYS
CHKDSK   .EXE      GRAPHICS.COM       RECOVER  .EXE
COMMAND  .COM      HIMEM    .SYS      REPLACE  .EXE
DEBUG    .EXE      LABEL    .EXE      RESTORE  .EXE
DISKCOPY.COM       MEM      .EXE      SETVER   .EXE
DOSKEY   .COM      MIRROR   .COM      SMARTDRV.SYS
EDIT     .COM      MODE     .COM      SUBST    .EXE
EDIT     .HLP      MORE     .COM      SYS      .COM
EMM386   .EXE      PRINT    .EXE      UNDELETE.EXE
FC       .EXE      QBASIC   .EXE      UNFORMAT.COM
                                      XCOPY    .EXE
```

11-1 MS-DOS stripped down to just the device drivers and utilities I use.

Of the roughly 84 DOS files (plus hidden files), I'm down to only 34—and that's for an 80386. For an 8088 or 286, the list of "keepers" would be even smaller. And if you have some pet 3rd-party goodies such as XTREE for a file manager or FASTBACK or one of the other glitzy backup packages, the list of what you really need to keep gets even shorter. The same for RAMDRIVE.SYS and SMARTDRV.SYS if you're using PC KWIK or some of the other alternates that are available.

When it comes to DOS externals, you need to keep only the ones you're actually going to use. All the rest is excess baggage. So look at Fig. 11-2 to see what you might want to dump. There are several files in this list with .CPI extensions. At most, only one or two of these is all you'll need, and maybe none at all. You might not even need one you might think you'd need. For example, there's an LCD.CPI you might think you'd need to run 5.0 on a laptop, but you do with only certain ones. My laptop runs just fine without it (or without any DOS screen driver). If you don't have a laptop or yours doesn't need this file, there's about an extra 10K you can pick up. If in doubt, put this one in the JUNKYARD.

11-2 As part of the trimming process, I was able to remove all of these video drivers and utilities from my VGA-equipped system.

```
DISPLAY SUPPORT:
4208        .CPI        720
5202        .CPI        395
EGA         .CPI     58,873
EGA         .SYS      4,885
GRAFTABL.COM        11,205
LCD         .CPI     10,753
MSHERC      .COM      6,934
4201        .CPI      6,404
```

And if you're using a VGA, you sure don't need over 60K of EGA support or any of those others device-specific items. I saved about 100K right here by dumping the files in Fig. 11-3.

11-3 Most users can recover another sizable chunk of disk space by deleting files used only when multilingual support is needed.

```
MULTILINGUAL SUPPORT:
DISPLAY .SYS     15,792
KEYB    .COM     14,986
KEYBOARD.SYS     34,697
NLSFUNC .EXE      7,052
PRINTER .SYS     18,804
COUNTRY .SYS     16,741
```

Here's another place most users can get rid of quite a load of excess baggage. For instance, at least one of my books is now available in Portuguese, but that doesn't mean I need to support a Portuguese keyboard and display on my machines—or any of another half a dozen languages I neither read or write in. So unless the work you do specifically requires the multilingual support that DOS provides, you can say goodbye to NLSFUNC, COUNTRY.SYS, DISPLAY.SYS, and KEYB. You don't even need to put these in temporary storage in the JUNK YARD. A tidy saving in any language.

There are, by the way, some other commands that enter into DOS's multilingual support, as well, but the ones listed in Fig. 11-4 are the only ones you either

```
MEMORY MANAGEMENT (Beyond 640K):
HIMEM   .SYS   11,488
MEM     .EXE   39,818
RAMDRIVE.SYS    5,873
SMARTDRV.SYS    8,313
EMM386  .EXE   91,742
```

11-4 Most 8088 owners can eliminate these files, although RAMDRIVE.SYS and SMARTDRV.SYS can be used in conjunction with 3rd-party EMS hardware and software driver, if available.

can (as in the case of CHCP) or would want to eliminate (as in the case of MODE .COM) because of other features not related to multilingual use.

Unless you're working with an 80286 or higher, you also don't have any need for DOS 5's memory management team—*not even if you have an expanded memory board installed in your machine*. This is because, among other things, EMM386.EXE, the DOS 5.0 EMS emulator, is for use *only* on 386 and higher systems, so even if you have a 286, scratch that one. If you have expanded memory on an 8088, just keep on using what you're using for an EMS driver because there's nothing here for you in DOS. And in that case, MEM.EXE is pretty much just excess baggage too.

Next, unless you have extended or expanded memory, you might as well forget about a disk cache (SMARTDRV.SYS) or a RAM disk (RAMDRIVE.SYS)—any cache or RAM drive, not just these. However, with a 286 (or up), unless you currently have some form of memory management, you should use HIMEM.SYS (assuming you have more than 640K on your machine), adding back the EMM386.EXE on a 386.

Table 11-1 lists still more files I eliminated and, in addition to these, if you don't want to use the shell there's about another half a megabyte just in the files in Fig. 11-5. Or if you don't plan to use EDIT and QBASIC there are the files in Fig. 11-6, totalling roughly another 400K of disk space.

```
DOSSHELL.COM     4,623      DOSSHELL.VID     9,515
DOSSHELL.EXE   235,380      DOSSWAP .EXE    18,724
DOSSHELL.GRB     4,421      DOSSHELL.INI    17,783
DOSSHELL.HLP   161,766
```

11-5 The 5.0 shell requires roughly 400K of disk space that can be recovered by anyone not using DOSSHELL.

```
QBASIC  .EXE   254,847      EDIT    .COM       413
QBASIC  .HLP   130,810      EDIT    .HLP    17,898
```

11-6 The new QBASIC interpreter is bulky and unless you're into BASIC—or want to use the new MS-DOS editor—here's another big chunk that many users can erase.

Table 11-1 A prize collection of excess baggage I elmiminated from my hard disk.

File	Size	Comment
APPEND.EXE	10,774	This one works backward; the function is better served by batch files.
ASSIGN.COM	6,399	Sometimes useful in the good old days, now it's a dinosaur.
COMP.EXE	14,282	Fielder's choice, but I like FC better.
DISKCOMP.COM	10,652	I've never needed this one, going back to DOS 2.0.
EDLIN.EXE	12,642	R.I.P.
EXE2BIN.EXE	8,424	Keep this one only if you're into programming.
EXPAND.EXE	14,563	Necessary only if you have to go back to the distribution disk.
FASTOPEN.EXE	12,050	Of limited benifit at best, and of no measurable benefit with caching.
FDISK.EXE	57,224	You could rarely use this from you hard disk anyway.
GORILLA.BAS	29,434	Keep it if you're into exploding bananas.
JOIN.EXE	17,870	Another dinosaur, of use only with floppy-only systems.
MONEY.BAS	46,225	A basic money and assets manager, some might find it useful.
NIBBLES.BAS	24,102	They should have quit with the exploding bananas.
PACKING.LST	2,538	Just a list of even more stuff than DOS put on your hard disk.
README.TXT	43,066	Read it first to see if any of it applies to you.
REMLINE.BAS	12,314	Removes line numbers from old BASIC/GWBASIC program files.
SHARE.EXE	10,912	Only used where files must be shared.
SORT.EXE	6,938	Usually better done by application programs.
TREE.COM	6,901	If you use DOSSHELL, you don't need this. If not, you might.

This isn't to say that I'm *recommending* you peel off this many layers of DOS—I'm just saying that these are possibilities you shouldn't overlook, if only because an extra megabyte of hard disk space bothers you. No matter how you slice it, megabytes cost money and I think I can invest my megabytes a little better. Especially as easy as it is to use EXPAND to bring back any space that you don't need now but might need later.

Part 4

Disk management

While disks coated with a thin coating of magnetic material are still the media of choice for most mass data storage, DOS 5 ushers in a new order. Beginning with the formatting of new media, the new DOS takes you where DOS has never taken you before—to a seemingly Utopian environment where lost files can be recovered, disks unformatted in case of accident, and data totally recovered from most hard disk crashes.

It still comes down to you, the user, to make it happen, however. So in this section I'll examine what's new and what's still unchanged—how to make DOS 5 work for you, but also how to cope with things like disk fragmentation and problems that just won't go away.

PART 4

Disk Management

CHAPTER 12

The new order

Disk management is peculiar in that you begin with chaos, impose a structured order on it, and let chaos triumph in the end—at least that's what too often happens, either through a lack of knowledge of the problems or inertia. Tomorrow is soon enough, you tell yourself—at least I do—but that, too often, is when yesterday and what you *should* have done comes back to haunt you.

In the beginning, there is chaos. The magnetic media you use, whether hard or floppy disks, are the epitome of chaotic perfection. And that has to be dealt with before you can even start to really mess things up—with a little help from DOS, that is.

Disk media: order out of chaos

As they come from the factory, disk media is perfectly blank—as nearly perfectly blank as is possible to manufacture. Some special high-density drives such as the three- and six-megabyte floppy disk drives formerly manufactured by Verbatim do require media with magnetic tracks that are prerecorded during manufacture, however, so there are exceptions. Other than these, there are no tracks or sectors, no marks of any kind—not even invisible magnetic markings—to define disks as even being disks. But they have to end up that way because, among other things, different operating systems use different schemes of disk data management. Apple computers, for example, can't read IBM-compatible disks, and try an Apple in an IBM and you'll get an *Invalid media error* message.

So before you can store data on disks, you must first impress some kind of structure on it, somehow break it up into address space that can be accessed with precision "on the fly"—and hard disks *do* fly. This is what disk formatting is all about—laying out that structure with an ordered series of magnetic check points.

5.0 introduces the most fundamental change this critical DOS utility has ever undergone. Through the years, DOS's formatting function was patched and bits and pieces added on as new disk formats were adopted, resulting in a hodge-podge of illogical parameters and switches. DOS 4.0 addressed the syntax problem, but this time around you have a whole new formatter that, for the first time (for DOS anyway), is nondestructive, i.e., it doesn't wipe out all the pre-existing data—even when you format floppies. Nondestructive formatters have been available for several years, most notably as part of the Paul Mace or Peter Norton utilities and more recently with PC Tools. This is the set piece for a whole new set of disk utilities in this release that I'll look at later in this section when I explain disk recovery. For now, I'm just going to concentrate on the basic job that must be done.

With DOS, all disks are laid out in a series of concentric rings, or *tracks*. An ordinary 360K 5¼-inch floppy typically has 48 tracks to the inch. The spacing must, obviously, be much tighter on hard disks. Tracks are then further divided into *sectors*, each sector capable of storing 512 bytes of data—a scheme that's essentially as shown in Fig. 12-1. On floppies, nine is the most common denominator in DOS, although DOS can also read, write, and format the eight sectors per track of 160 and 320K drives, more for high-density drives.

These tracks and sectors are laid out in the form of a circular magnetic gridwork pattern during the process of *formatting*. It isn't enough to merely lay out and mark the grid "lines," so to speak, however. Each sector has to be given a specific address or number for later reference.

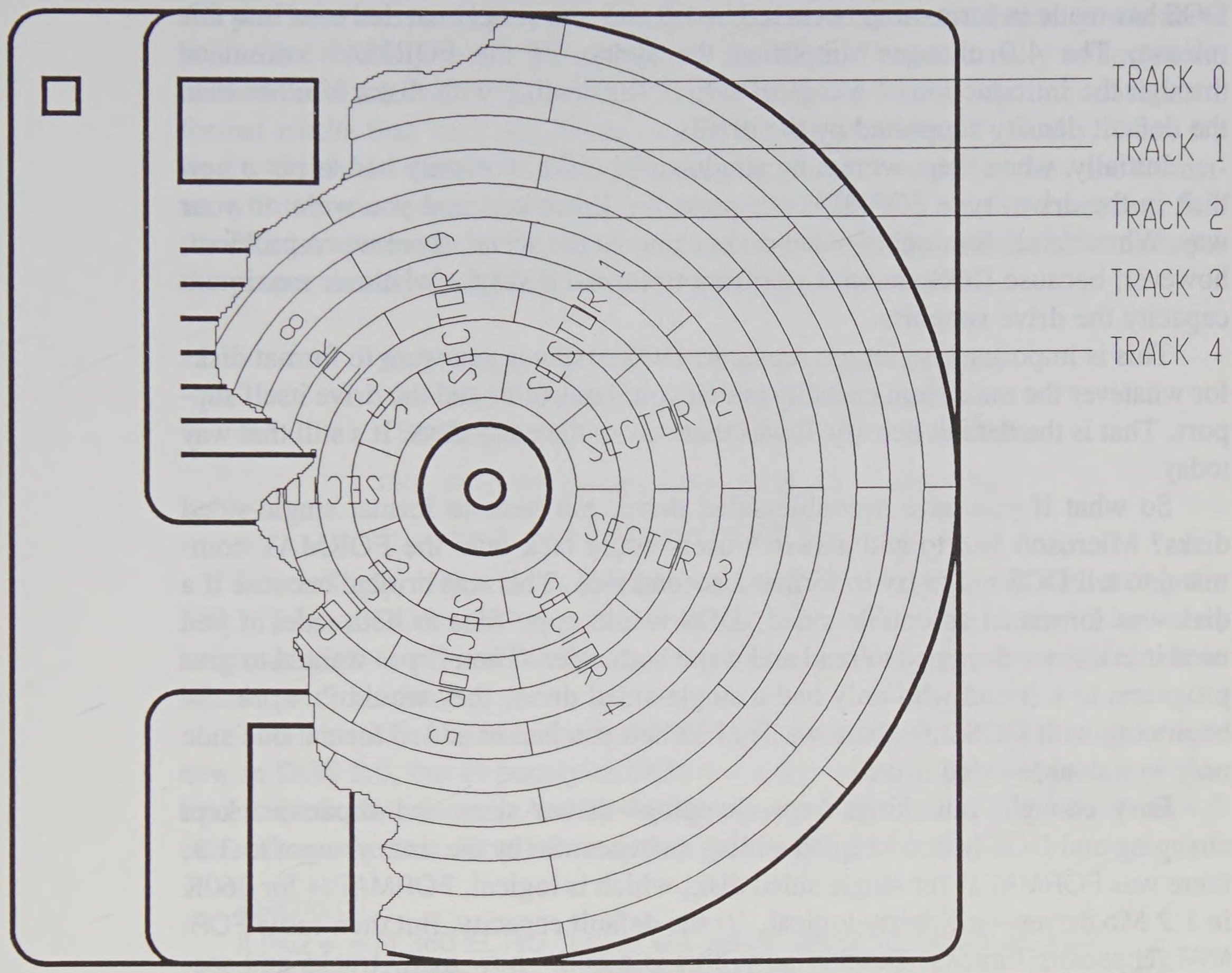
This is true even in the case of certain software—including DOS's DISK COPY.COM—that can work with previously unformatted disks. Such software must actually format "on the fly," creating tracks and sectors exactly as if it were formatting. In any event, it's a pretty complex process—much more so than the average user appreciates.

It's not enough, however, to simply define the layout of a disk, provide space for directories, etc. A key to how the disk is actually laid out must be written to one sector, called the *boot sector*, so DOS can recognize the way that the disk is formatted—whether it's 360K or 1.2 Meg, in the case of 5¼-inch floppies, for example—and knows just where to look to find the root directory and the FAT.

Disk space: lost and found

During the formatting process, you inevitably lose something. Typically, a floppy disk advertised as a 2 megabytes holds only 1.44 Mb, etc. What happened?

Both numbers are right. The larger number is the *unformatted* capacity—



12-1 Cutaway of typical floppy disk showing the format as a series of concentric rings (tracks) with each track divided into sectors. Each sector can hold 512 bytes of data.

that's how much data the disk can actually hold. The smaller number is how much space is left for you to store your data after DOS has scattered crumbs of its own data to mark a trail that it can follow to any sector. Each sector actually contains more than the 512-byte area that you can use. The rest is taken up with housekeeping data required by DOS. You don't see it, but it's there.

Clearly then, whether you're talking floppy disks or hard disks, this formatting process is one of the most critical functions your operating system has to perform, not just once but every time you introduce it to a previously unused or recycled disk. This is the foundation of your whole data storage scheme. An error here can bring the entire system tumbling down like a house of cards. It's very crucial!

From the standpoint of most users, probably the most important change that

DOS has made in formatting occurred in 4.0 and was simply carried over into this release. The 4.0 changes simplified the syntax of the FORMAT command through the introduction of a *logical* system for dealing with disks of other than the default density supported by the drives.

Initially, when there were only single-sided disks, you only had to put a new disk in the drive, type FORMTAT *d:*, press the Enter key, and you were on your way. When drives for double-sided disks came on the scene, there was a problem, however, because DOS *assumes* you want to format disks for whatever maximum capacity the drive supports.

This is important, so let me repeat it: DOS *assumes* you want to format disks for whatever the maximum capacity is that your controller and the drive itself support. That is the default density. It was that way in the early days. It's still that way today.

So what if you have a double-sided drive, but have to format single-sided disks? Microsoft had to add a *switch* users could tack onto the FORMAT command to tell DOS not to try to format a second side. This was critical because if a disk was formatted as double sided, DOS would copy files to both sides if you used it in a drive designed to read and write both sides. Then if you wanted to give programs to a friend who only had a single-sided drive, they wouldn't work. So beginning with DOS 2.0, there was a /1 switch you had to add to format one side only in a double-sided drive.

Easy enough, but things kept changing—drives sizes and capacities kept changing and DOS had to keep on adding switches. So by the time you got to 3.3, there was FORMAT /1 for single sided disk, which is logical, FORMAT /4 for 360K in 1.2 Mb drives—still fairly logical, 1/4 the default capacity. But then came FORMAT /N:*sectors*/T:*tracks*. Quickly now, this is a quiz—how many tracks and sectors are there on *your* floppy disks? I doubt if one in a hundred users could answer that, nor is there any reason why they should have to.

You still had /4 to format 360K disks, but for those (then) new-fangled 3 1/2-inch drives—and presumably whatever else was coming down the pike—you had to know how many tracks and sectors DOS should format for.

One of the few things DOS did right in 4.0 was, while still supporting the old switches for the convenience of users who had learned them, to add a new switch called the *form factor*. For any other than the default drive capacity, you simply specified how many *bytes* you wanted—something any user should surely know:

```
FORMAT /f:360
FORMAT /f:720
```

And DOS won't let you just fill in some odd-ball figure. Drives can format only certain sizes, and if the size you call for isn't one of them the formatter reports:

```
Parameters not compatible
Format terminated
```

All you have to remember is to specify a different formatted capacity whenever you're not working to the full capacity of the drive. There's usually no long-term harm done if you don't. The drive will groan a lot and take a long time as it tries to format media that isn't suitable to handle higher densities. Typically, when it's finally finished—usually several minutes longer than it ought to take—the formatter will tell you something like what's shown in Fig. 12-2. In this case, over half the disk is reported to be bad sectors. In most cases, the disk can be reformatted to the proper density (by giving the correct FORMAT command).

```
1213952 bytes total disk space
 675840 bytes in bad sectors
 538112 bytes available on disk
```

12-2 DOS generally sees a lot of bad sectors if you're trying to format a disk to a higher capacity than it was intended for, as with this ordinary 360K DSDD floppy.

Interestingly, if you're in a bind, you could probably get by with using the 538,112 bytes DOS says are good. I've had to once or twice. I don't, however, recommend it—and particularly not in any situation where an error could result in loss of precious data.

This simplified syntax for the FORMAT command, as stated earlier, isn't new in DOS 5.0, but as poorly as DOS 4.x was received, this change has probably escaped a lot of users. And by the way, users of versions earlier than DOS 4.0 can still use the new form factor syntax with this little batch file:

```
@echo off
If !%2 = = !/f:360 ECHO Disk in %1 will be being formatted as 360K
If !%2 = = !/f:360 [path\]FORMAT %1 /4
If !%2 = = !/f:720 ECHO Disk in %1 will be being formatted as 720K
If !%2 = = !/f:720 [path\]FORMAT %1 /n:9/t:80
If !%2 = = !/f:720 ECHO Disk in %1 will be being formatted to default capacity
of drive
If !%2 = = ! [path\]FORMAT %1
```

Only the lines shown in bold print are required to make this work, although @echo off is recommended to keep the screen from being cluttered. The remaining lines are entirely optional, doing nothing but prettying up the presentation. If you use the entire batch file as it's presented above, make sure *not* to break the line that contains the *formatted to the default capacity of drive* line with a carriage return (broken here because of the line length), because DOS will mistake anything following the Return as as separate command. Also, use blank spaces *only* where shown.

And in all of this, I've barely even mentioned what the big news is—the nondestructive default floppy formatting capabilities. This really is the big news, and

I haven't ignored it to belittle its importance. However, a nondestructive format becomes important *only* if you suddenly discover you've screwed up and formatted the wrong disk—or at least before you realized it still had files or data that you desperately need. And I'll talk about that in detail, later in this book in chapter 15.

Fragmentation: chaos out of order

Unfortunately, DOS has some peculiar habits in the way it goes about storing data on disks. The face you see when you call for a directory listing—even the unalphabetized listings prior to 5.0—bears little resemblance to the systematic chaos DOS makes of your disk. Disks coated with a thin coating of magnetic material are still the media of choice for most mass data storage. They're the only ones you need concern yourself with here, and to understand DOS 5.0's new disk management capabilities you need to understand a little bit about the media it has to manage. No matter how carefully you think you've organized your files into directories and subdirectories, left on its own, DOS will start scattering odd bits and pieces of files to the four winds.

At very best, this process, called *disk fragmentation*, slows you down because DOS has to search for all those scattered bits and pieces every time you try to load a program or even just access a file. At worst, it makes it difficult if not impossible to recover data should some accident occur.

While this all seems quite mysterious at first glance, it really isn't if you stop to study it. More important, though, with a little understanding there's much you can do about it. You can head such problems off before DOS really gets a chance to make a mess of your disk, or even put things back in order if they've already gotten out of hand. It's never too late—at least not until that ultimate catastrophe occurs.

Even then, if you've used 5.0's MIRROR to save a copy of the data in the File Allocation Table and the root directory (and preferably the partition table, too) the likelihood of effecting a complete recovery is pretty high.

But that still leaves you with a hard disk that runs slower and sloowweerr and sllooowwwrrrr—more than ample reason in itself to nip the problem of file fragmentation in the bud.

DOS, the scavenger

It all comes down to one simple fact. DOS is a scavenger. DOS was heavy into recycling before recycling became a buzzword. It *has* to be a scavenger. If it weren't, you could use disks only once, regardless of whether the files on them were any good or not.

Starting with a new or freshly formatted disk, DOS copy files sequentially,

using as many sectors for each as needed, in this manner, with a single file represented by a series of the same number or letter:

```
0001122223445555555566677777778999999999AAAAABBCCCD...
```

As a quick aside, this is exactly the way DOS lists files in the root directory: in the order in which they were copied to the disk. And this is the way DOS has always shown them in directory listings up until now—the order in which they were copied to the disk, at least up to the point where DOS has to start recycling root directory entry space. There's a logic to everything in DOS of you dig deep enough. Well, almost everything. Somebody's logic anyway.

Back to the way DOS stores your files on disks—at some point you start erasing old files, so your disk begins to look like this:

```
000ee222234455555555eee7777777e999999999AAAAAeeCCCD
EEEEEFFFF
```

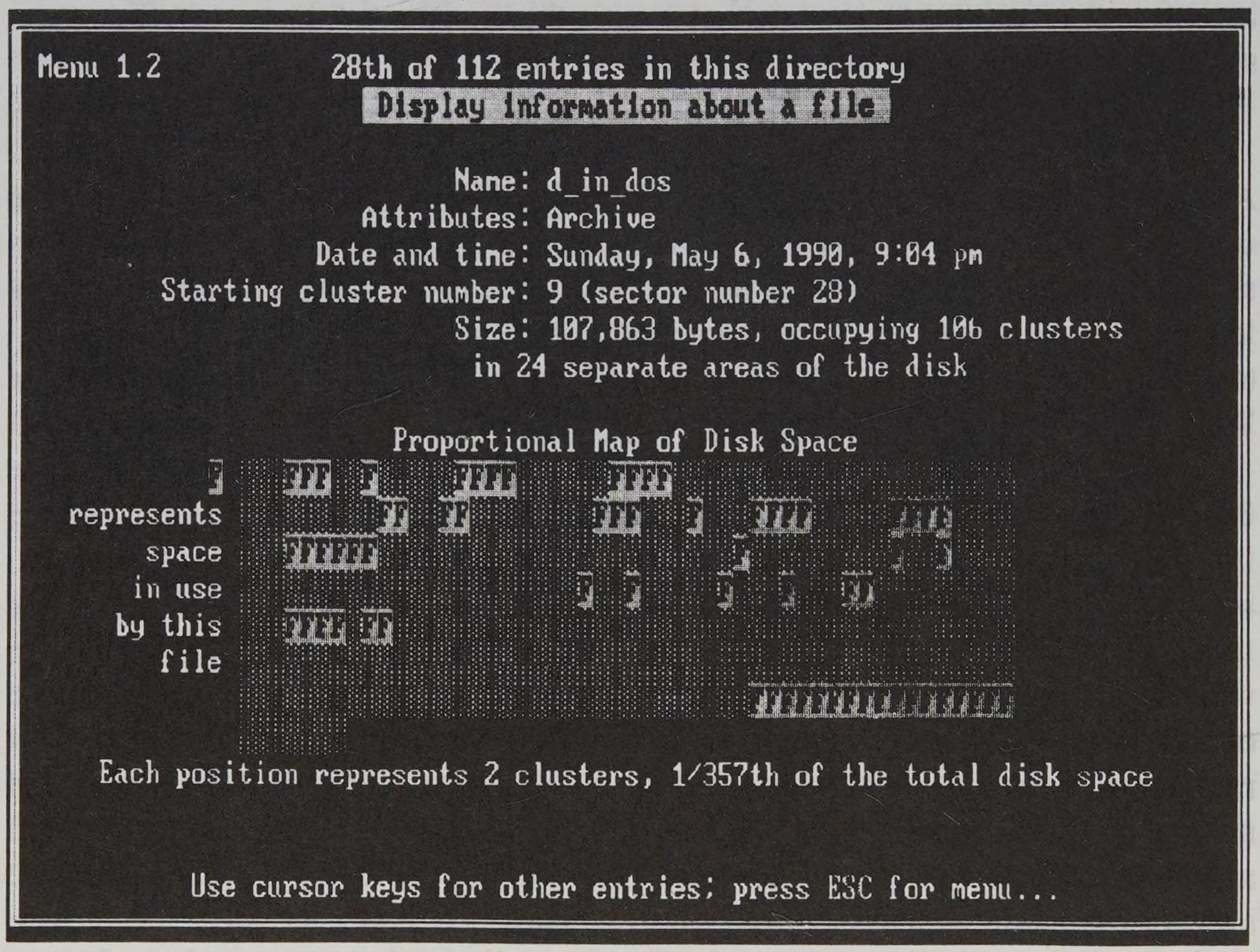
with the *e*s representing free disk space. It still might come as a surprise to some of you to know that all those files you've "erased" from your disks have never really been erased. All DOS does is alter the directory entry in such a way that it no longer shows up when you give a DIR command. That, and mark the disk space the data occupies as space DOS can reuse (write over) when it needs the space for something else.

That "erased" file space might remain in limbo for some period of time. DOS knows it's there—it keeps careful inventories of its resources. But it won't reuse the space as long as there's fresh, unused file space still remaining on the disk. This is important because, as long as DOS hasn't actually written over the original data with new files, it's usually possible to recover the original files intact with the new DOS 5.0 UNDELETE utility, discussed elsewhere in this book.

Since version 3.0, DOS has held erased filespace open as long as possible before writing over it. "Erased" space also includes space that your software has abandoned (like old .BAK files that newer .BAK files have replaced). Prior to that, DOS wrote over the most recently abandoned filespace first, making the recovery of inadvertently erased files pretty iffy, even if you had the Norton Utilities at hand to work with. As file recovery became more viable, a whole genre of unerasing and data recovery software was developed by 3rd-party sources through the years.

But let's get back to your imaginary disk. All the file space has been used the first time around and now you have another file you want to copy to it—a fairly big one. DOS checks its resource inventory and finds it has enough erased file space available—the spaces marked with *e*s above—and it obediently copies your new file, represented as a series of Xs, into those old spaces this way:

```
000XX222234455555555XXX7777777X999999999AAAAAXXCCC
DEEEEEFFFF
```



12-3 A Norton Utilities screen of a portion of the FAT, showing a large fragmented file.

And Fig. 12-3 shows an analysis of an actual disk that has a rather large fragmented file. This, by the way, isn't a problem that's unique to DOS.

What to do about fragmentation

There are several ways of dealing with the problem of fragmentation. You can:

- Ignore it and see disk performance get slower and slower, also increasing the likelihood of irretrievable data loss in case your disk should suffer even some relatively minor mishap.
- Recopy any files that you suspect DOS might have fragmented to fresh disks.
- Reformat your disk periodically to give DOS a clean slate, so to speak.

Most people would agree that the first option is unacceptable. And obviously the last two, while fine for floppy disks, aren't feasible when dealing with a hard disk. So I'm now going to do something I've rather carefully avoided doing in this book—discuss something that DOS doesn't support, not even in this release: unfragmenting or "optimizing" disks.

DOS doesn't support unfragmenting. They do, however, mention it in several places in the DOS 5.0 documentation, and it's a well proven technology—demonstrated not only to be safe but to significantly speed up disks operation.

Mass migration

Unfragmenting broken files means moving them. This process should not be confused, however, with moving files between directories on the same disk as some 3rd-party utilities allow by simply moving their directory entries rather than the actual data sections. And, although primarily for hard disk use, this technique can be used with floppy disks, as well (although there isn't usually much of a need to do so).

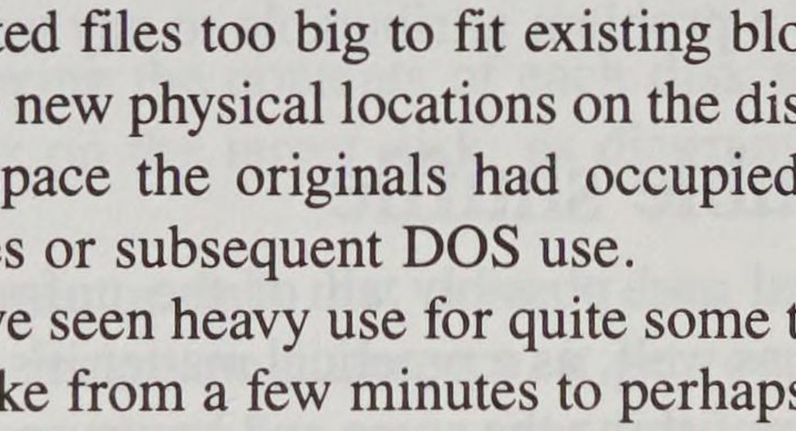
Programs that unfragment disks begin by sorting out which files are fragmented and cataloging them. Then they look for *contiguous* blocks of available space they can be copied into. Moving smaller files is fairly easy, and every reasonable effort is made to match file sizes neatly to the blocks they're moved to, as shown in Fig. 12-4, where a fractured file that takes up five allocation units—the smallest block of space that DOS will deal with on a disk—is moved into a slot just big enough but not too big.

Here, a larger block is ignored to achieve a perfect fit. It's sort of like a giant jigsaw puzzle. Some of the more sophisticated unfragmenting algorithms might actually move files that aren't fragmented sometimes to open up blocks big enough to accommodate unfragmented files too big to fit existing blocks.

Once files have been copied into new physical locations on the disk, the original files are erased and whatever space the originals had occupied is released either for the relocation of other files or subsequent DOS use.

In the case of hard disks that have seen heavy use for quite some time without unfragmenting, the process might take from a few minutes to perhaps a half hour or more. If repeated on a regular basis, however, say once a month, the process generally requires just seconds to complete.

It should be noted, though, that the process of unfragmenting generally puts an end to any chance of undeleting any old "erased" files that might have survived



00000eeeeeee2222223333333eeeee555555XX7777XXX99999999

12-4 A disk unfragmenter works by gathering scattered bits of files, copying them to empty blocks big enough to hold the whole file, and then erasing the original.

up to that point. Like most things, this is a trade-off, but the long term benefits usually outweigh any disadvantage.

Optimizers I've used extensively and on a regular basis over the past several years include the two following:

VOPT
Golden Bow Systems
P.O. Box 3039
San Diego, California 92103
(800) 284-3269

OPTUNE
Gazelle Systems
42 N. University Ave., Suite 10
Provo, Utah 84601
(800) 373-6933

There are others that are probably as good but both of these, while possibly not as neat and thorough as, say the one included with the Peter Norton Utilities, are fairly fast, typically requiring only a few seconds if done on a regular basis. Optune, however, offers several levels of unfragmentation, allowing for a quick job, say, on a daily basis but a more thorough tidying-up on a more infrequent basis.

Caution: While unfragmenting is a proven technology and is generally considered safe, it does involve the manipulation and relocation of data and as a precaution all critical files and data should be backed up before unfragmenting. Through several thousand unfragmentation cycles using several different optimizing programs on a variety of hard disks over several years, the authors have never encountered a problem attributable to any unfragmentation operation.

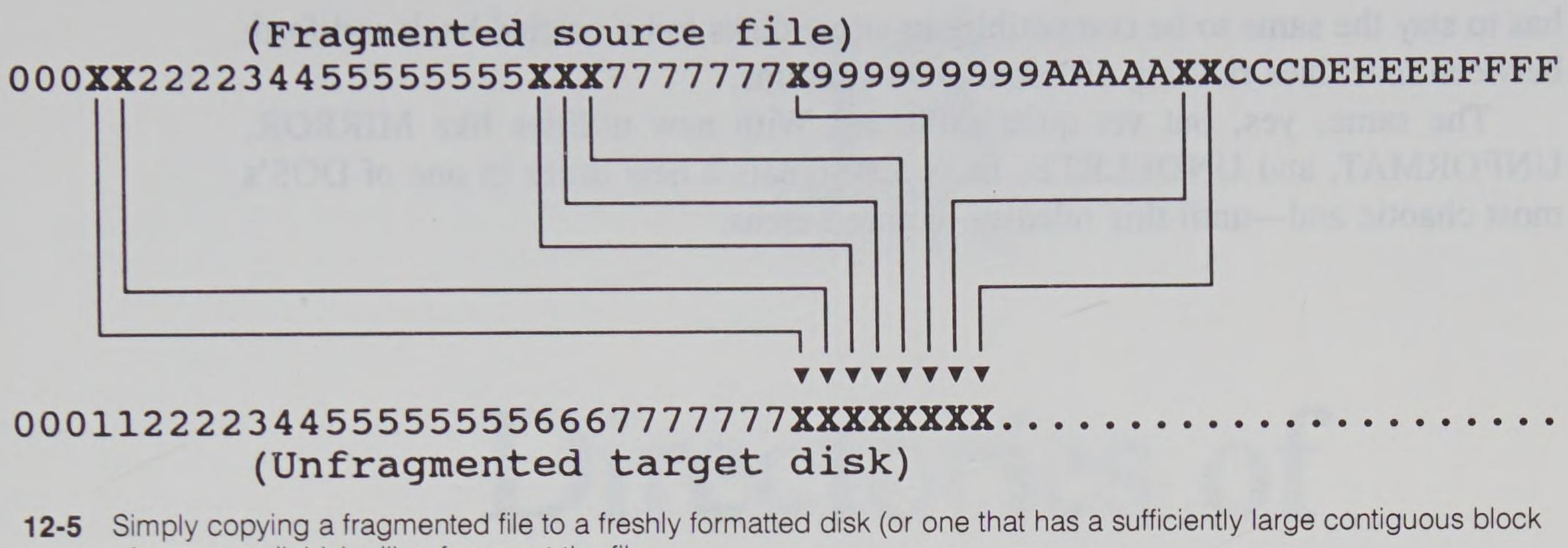
The double shuffle

While several and possibly all of the unfragmentation utilities will work with floppy disks as well, as a practical matter it's generally not done. There is another way of accomplishing the same end however, even in the case of badly fragmented source files. DOS, you see, only starts fragmenting files where it finds obstacles standing in the way of writing to consecutive spaces.

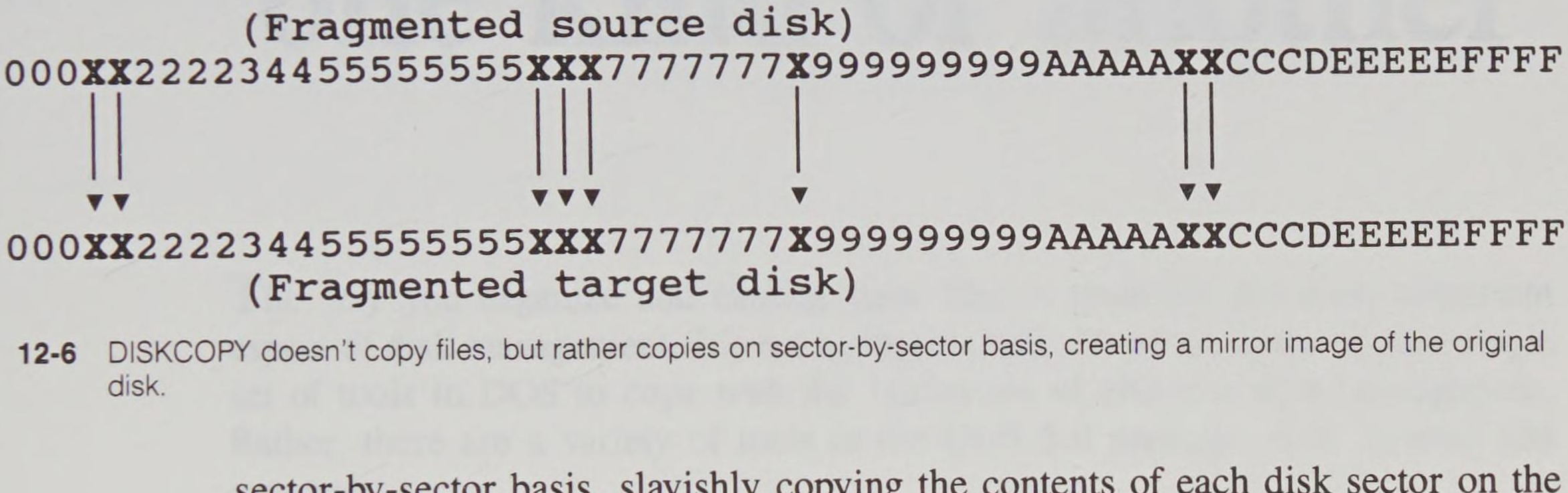
Shown in Fig. 12-5, then, is an illustration of the same fragmented file in Fig. 12-4, simply copied to a floppy disk that still has enough filespace unused to fit the file in one chunk.

COPY does it, and XCOPY does it. In practical terms, that means it's not a bad idea to periodically copy any floppy files you want to save to freshly formatted disks and then reformat—not just erase—the old disks. That way you'll start off with a clean slate when you reuse them. The fact that the new 5.0 FORMAT is nondestructive doesn't really matter. It does zero-out the FAT and root directory areas, however, which is where the seeds of file fragmentation are sown.

Unlike COPY and XCOPY, DISKCOPY doesn't unfragment fragmented files. It doesn't even try because it doesn't copy files, it only copies disks on a



12-5 Simply copying a fragmented file to a freshly formatted disk (or one that has a sufficiently large contiguous block of space available) will unfragment the file.



12-6 DISKCOPY doesn't copy files, but rather copies on sector-by-sector basis, creating a mirror image of the original disk.

sector-by-sector basis, slavishly copying the contents of each disk sector on the source disk to the exact same sector on the target disk, as diagrammed in Fig. 12-6.

DISKCOPY, in fact, will copy sector data that doesn't even match a valid entry in the FAT or root directory. It makes no difference, it gets copied anyway. In an extreme case, if you want to have a little fun some time just load a disk with files—something you don't care about. Erase the whole disk and then DISKCOPY the "erased" disk to another disk. Now you've got two "blank" disks. However, if you then run UNDELETE on the "blank" target disk, you'll find you that you can actually undelete all the files you had on the source disk before you deleted them. DISKCOPY has faithfully copied not only all the erased data, but all the erased FAT and root directory entries, as well. It's kind of weird, but it works.

To put things in perspective, you must keep in mind that disk fragmentation isn't something new or even unique to DOS. It's the same in OS/2 and a necessary evil with virtually any operating system that recycles disk space on reusable media.

Nor is there anything that's novel in the way the new DOS formats disks, except that it's nondestructive. But the actual DOS format, the structure DOS imposes on the disk, is no different today from what it was in earlier releases—it

has to stay the same to be compatible, to allow disks to be carried back and forth between machines running different DOS versions.

The same, yes, but yet quite different. With new utilities like MIRROR, UNFORMAT, and UNDELETE, DOS 5.0 signals a new order in one of DOS's most chaotic and—until this release—ignored areas.

CHAPTER 13

Directories of one kind or another

The way you organize and catalog your files is probably the most important aspect of disk management. It's not a singular task, however, nor is there a single set of tools in DOS to cope with the intricacies of effective disk management. Rather, there are a variety of tools in the DOS 5.0 package, both internal and external.

Other than the file management features of the new DOSSHELL, certainly the most significant advance in the area of file management, is the greatly enhanced DIR command, which adds seven sorting options beyond the default alphabetical sort finally introduced with version 4.0.

Starting with DOS 5, DOS will sort directories almost any way you want them sorted—and by default! You set whatever default suits your needs and DOS does it. I usually want mine alphabetical with files and subdirectories listed separately, and with the display pausing after every screen. And that's *exactly* what DOS 5 gives me automatically, thanks to a variable set in the environment by the AUTOEXEC.BAT—I never even have to give it a thought.

For the times I need a different sort order or need some different display format, all I need to do is add switch(es) to the DIR command on the command line, like this:

DIR dos/w/p/o and Enter

for a one-time-only different kind of sort, like what's shown in Fig. 13-1. I'll bet you've never seen anything like that from DOS before. The new sort options are

```
 Volume in drive C has no label
 Volume Serial Number is 1677-874F
 Directory of C:\DOS

[.]                 [..]                [386MAX]            [4DOS]              [ALL]
[BALL]              [BATCH]             [BUFFALO]           [COMMANDR]          [DOS]
[DOS_HIDE]          [DOS_JUNK]          [DV]                [FICTION]           [FORMWORX]
[JUNKYARD]          [MEMO]              [NAMEDISK]          [PCKWIK]            [PROCOMM]
[PROJECTS]          [PZP]               [P_CARD]            [QEMM]              [QUICKEN]
[QUICKEN4]          [SCAN]              [SIDEKICK]          [SITBACK]           [SQUISH]
[STAK]              [SUPRCALC]          [TAB]               [TC!]               [TEMP]
[UTILITY]           [VM386]             [VM386_MU]          [XTREE]             [XYWRITE]
ATTRIB.EXE          BACKUP.EXE          CHKDSK.EXE          COMMAND.COM         CURRENT.BAT
DEBUG.EXE           DISKCOPY.COM        DOSKEY.COM          EDIT.COM            EDIT.HLP
EMM386.EXE          FC.EXE              FIND.EXE            FORMAT.COM          GRAPHICS.CO
GRAPHICS.PRO        HIMEM.SYS           LABEL.EXE           MEM.EXE             MIRROR.COM
MODE.COM            MORE.COM            PRINT.EXE           QBASIC.EXE          QBASIC.HLP
RAMDRIVE.SYS        RECOVER.EXE         REPLACE.EXE         RESTORE.EXE         SETVER.EXE
SMARTDRV.SYS        SUBST.EXE           SYS.COM             UNDELETE.EXE        UNFORMAT.CO
XCOPY.EXE
       82 file(s)     1095126 bytes
                      2967552 bytes free
Press any key to continue . . .
```

13-1 Wide display with the new 5.0 DIR command, showing directories and files listed separately, both in alphabetical order.

based on the following:

/o:n	A-Z alphabetically (the default)
/o:e	A-Z by extension
/o:s	Smallest first
/o:d	Oldest first
/o:g	Directories before files

There's something to watch out for here, though. While the 5.0 default is alphabetical, it's *totally* alphabetical, with directories and files mixed together in whatever order the alphabet decrees. Obviously, as you can see from the directory in Fig. 13-1, there's a way around that though: the /o switch used without any modifier. That's the trick to putting alphabetized directories first, followed by alphabetized files.

Another first for the DOS DIR command is its ability to display files according to which attribute bits are set, with the following switches:

/a:a	Displays only if archive bit is on
/a:r	Display is limited to read-only files
/a:h	Displays only hidden files
/a:s	Displays only system files

And, lumped with the attribute selectors, there's also a switch for displaying directories:

/a:d	displays only directories

The meaning of any of these can be reversed by simply putting a minus sign ahead of the designating letter: −n to have the directory sorted from Z−A, or −d to exclude directories and show files only, to name two.

Carried over from earlier versions are the /p (pause to present only one screenful at a time rather than scrolling to end) and /w (display an abbreviated directory in five columns as, filename and extension only sorted from left to right). There are also three more new switches worth noting:

/s	Files in specified directory and all subdirectories
/b	Bare format (no heading information or summary)
/l	Displays in lowercase

I find /b especially handy for sorting and gathering files into a list for chain printing through my word processor. This command, for example, can create a list for printing the files of an entire manuscript in the correct sequence.

```
DIR V:\FINISHED /b/o:e >e:book.doc
```

Switches, as you've probably noticed, can be combined, although I've found that some combinations will shoot each other down, with the last one entered on the command line prevailing.

The key to really unlocking the power of the newly enhanced DIR command is a new environmental variable, DIRCMD. You can include a SET command in your AUTOEXEC.BAT or from the command line if you prefer. I use the following line in my AUTOEXEC.BAT file—right after the PROMPT line:

```
SET DIRCMD /o/p
```

You can override this setting on the command line at any time on a one-shot basis—for either the remainder of the session or until you change your mind again—by entering a new SET DIRCMD = */xxx* command.

Attribute bits

Interestingly, although DIR now allows you to sort files according to the way their attribute bits are set, for all of the options none will actually *show* the attribute bits. There's now a way around that, however, with DIR. You'll have to use the ATTRIB command to see what's going on. Entered without parameters (except drive and path as required) the ATTRIB utility displays something like a directory listing—like the old *unsorted* DOS directory listings—showing the attribute status but no file size or time and date stamp data, as shown in the excerpt in Fig. 13-2.

The ATTRIB utility itself has been considerably enhanced in 5.0. While previous releases have displayed the status of all four attribute bits, until now DOS has allowed no user access to the H (hidden) and S (system) attributes, limiting tampering to the more benign R and A bits.

```
   SH      C:\IO.SYS
   SH      C:\MSDOS.SYS
           C:\AUTOEXEC.BAT
A          C:\COMMAND.COM
A  SHR     C:\MIRORSAV.FIL
     R     C:\OLD.C
     R     C:\WINA20.386
A    R     C:\MIRROR.BAK
A    R     C:\MIRROR.FIL
```

13-2 Although the new 5.0 DIR command can limit display to only files with certain attributes, in order to see which bits are set you still have to use the ATTRIB utility, as shown here.

There are few occasions, however, when it's desirable, let alone necessary, to fiddle with the S and H bits—although I don't advise it unless you have good reason or are reasonably experienced in such matters. This is particularly true of switching those attributes off on IO.SYS and MSDOS.SYS.

You can, of course, hide things like COMMAND.COM if you like playing games and want to reduce the clutter in your directory. For that matter, you can hide almost any files you like and still call them up the same as if you hadn't hidden them. About the only difference is that they won't show in a directory listing. At least you could before. With the new DOS 5, all bets are off on that score.

The R (read only) attribute is one you should be familiar with on a working basis. It's a good way to significantly reduce the risk of accidental damage to *any* files you want to retain in their original (or current) state. This, of course, is invaluable where multiple users have access to certain data but not reason to alter that data.

It can also be extremely useful even in a single user system, however, for files used as templates for the creation of other files—a spreadsheet listing students names and other data, for example, that can be used as a basis for creating a whole set of spreadsheets. The seconds it takes to type ATTRIB +r *filename* is time well used.

The eloquent tree

"A poem lovely as a tree," someone once waxed eloquent. Clearly, the poet had never seen some of the hard-disk trees I've seen, or the way DOS shows them with TREE. That the trees themselves are often tortured things is through no fault of DOS; the way DOS shows them to us is an entirely different matter.

The TREE command has been around virtually unchanged since the DOS 2.x era. The way it actually displayed changed in version 3, but this was cosmetic. And at some point a /a switch was added to replace the extended character-set characters used by default with pure ASCII characters, which allowed you to get a reasonable printout if you redirected the output to a printer that didn't support the extended character set.

```
|                                  3                                  |
+---DOS                            CDDDDOS                            +---DOS
|   |   FORMAT.COM                 3    3    FORMAT.COM               |   |   FORMAT.COM
|   |   GRAPHICS.PRO               3    3    GRAPHICS.PRO             |   |   GRAPHICS.PRO
|   |   HIMEM.SYS                  3    3    HIMEM.SYS                |   |   HIMEM.SYS
|   |   MODE.COM                   3    3    MODE.COM                 |   |   MODE.COM
|   |   SETVER.EXE                 3    3    SETVER.EXE               |   |   SETVER.EXE
|   |   DEBUG.EXE                  3    3    DEBUG.EXE                |   |   DEBUG.EXE
|   |   DOSKEY.COM                 3    3    DOSKEY.COM               |   |   DOSKEY.COM
|   |   EMM386.EXE                 3    3    EMM386.EXE               |   |   EMM386.EXE
|   |   MEM.EXE                    3    3    MEM.EXE                  |   |   MEM.EXE
```

13-3 Excerpt from a TREE display. On the left is the default screen display, in the center the way the display reproduces on a printer that doesn't support the high character set, and at the right the same printout as the middle, but with the /a switch.

As shown in Fig. 13-3, on the left you have a sample of the default display, in the center, as printed on a printer that supports only the low character set and, at the right, the way that portion prints on the same printer when the /a switch is used.

Like DIR, however, TREE is a one-way street. It scrolls only from top to bottom. Lacking still is any way to scroll back up again, to browse or mark files for deletion, or to move or copy them. This is still just basic, bare-bones DOS—hardly at its finest.

CHKDSK with the /v (verbose) switch also does a complete listing of all the files and directories on a disk, but this is one I'd just as soon forget.

New kid on the block

The DOSSHELL is discussed in greater detail in chapter 16, but it seems appropriate to look briefly here at some of its file management features, alongside those available from the command line. The excerpt shown in Fig. 13-4 is from a file list display in text mode. Though not totally new in this release, the shell clearly adds a new dimension to the file management capabilities of DOS.

```
                                MS-DOS Shell
 File  Options  View  Tree  Help
 C:\DOS
[A:]  [B:]  [C:]  [D:]  [E:]  [F:]  [I:]  [J:]  [L:]  [M:]  [O:]  [P:]  [Q:]
[R:]  [S:]  [T:]  [U:]  [V:]  [W:]  [X:]
       Directory Tree                          C:\DOS\*.*
 [-] C:\                                ↑     ATTRIB   .EXE      15,796  03-08-91
   ├─[-] DOS                                  CHKDSK   .EXE      16,200  03-08-91
   │   ├─[ ] 386MAX                           COMMAND  .COM      47,867  03-08-91
   │   ├─[ ] 4DOS                             CURRENT  .BAT         278  04-25-91
   │   ├─[ ] ALL                              DEBUG    .EXE      20,634  03-08-91
   │   ├─[ ] BALL                             DISKCOPY.COM       11,793  03-08-91
   │   ├─[-] BATCH                            DOSKEY   .COM       5,883  03-08-91
   │   │   └─[ ] DEMO                         EDIT     .COM         413  03-08-91
   │   ├─[ ] BUFFALO                          EDIT     .HLP      17,898  03-08-91
```

13-4 Portion of a DOSSHELL directory and file listing.

At first glance of the figure, you can see what disk resources are available, including substituted drives (and, as you can see, I routinely use a lot of substituted drives). Most noteworthy is the alphabetized display of directories in the drive currently being read, down the left side of the screen, and the files in the drive currently selected (highlighted on the screen), down the right. While this is nothing several 3rd-party vendors haven't been selling profitably in the aftermarket for several years, it *is* new to DOS. Unlike TREE or DIR, the DOSSHELL display allows you to browse back and forth at will. So now we're getting somewhere . . . almost.

Like most shell features, this one was written primarily with mouse users in mind and is surely faster with a mouse than from the keyboard. However, having said that, DOSSHELL is at best considerably slower to work with than, say, XTREE—even with one mouse tied behind your back.

I found a number of the features to be clumsy, anyway you slice it. For example, to find out what attribute bits are set, you have to go to the All Files mode, which results in a display similar to the one in Fig. 13-5—literally *all* the files on the disk; almost 1500 in this case. And then All Files doesn't include hidden files, so it really isn't all files. Score that round for ATTRIB. And if you hide a file, forget it. The shell won't display a hidden file. While playing with it, I hid one and consequently lost it completely until I exited to the command line and ran ATTRIB -H *filename*—sort of like taking off a voodoo curse.

```
                                MS-DOS Shell
 File  Options  View  Tree  Help
 C:\DOS\386MAX
[A:]  [B:]  [C:]  [D:]  [E:]  [F:]  [I:]  [J:]  [L:]  [M:]  [O:]  [P:]  [Q:]
[R:]  [S:]  [T:]  [U:]  [V:]  [W:]  [X:]

                                                        *.*
 File                                    EMM386   .EXE      91,742  03-08-91   5:05a
   Name   : EMM386.EXE                   EMS      .COM       6,027  09-07-90   5:11a
   Attr   : ...a                         EMS      .SYS       6,209  09-07-90   5:11a
 Selected                        C       EMS2EXT  .SYS       3,040  09-07-90   5:11a
   Number:                       1       ENVELOPE             37  09-27-90   1:51a
   Size   :                91,742       EPSAMBER.PRN       3,105  11-18-86   7:58p
 Directory                               ERR      .BAT       1,686  12-19-90  11:55a
   Name   : DOS                          ERRMAXIM.BAT         279  02-17-91  10:10a
   Size   :             1,044,221        ERROR    .BAT      14,278  04-04-90  11:19p
   Files  :                    36       ESC      .BAT          16  11-24-90  12:51p
 Disk                                    ESC      .SCR          41  11-24-90  12:42p
   Name   : none                         EXE      .BAT           5  10-08-90   4:27p
   Size   :            33,396,736        EXE      .BAT           5  10-08-90   4:27p
   Avail  :             2,985,984        EXE2BIN  .EXE       8,424  03-08-91   5:05a
   Files  :                 1,492        EXP      .BAK      12,758  04-14-91   8:02a
   Dirs   :                    68        EXP      .KBD      13,033  04-14-91   8:27a
                                         EXPAND   .EXE      14,563  03-08-91   5:05a

F10=Actions  Shift+F9=Command Prompt                                        5:59p
```

13-5 The Show Information option displays disk and file specifics when using the shell for file management.

The shell does allow you to selectively mark individual files for group processing. Again, I found this to be a little clumsy compared to some of the top 3rd-party shells, but it gets the job done.

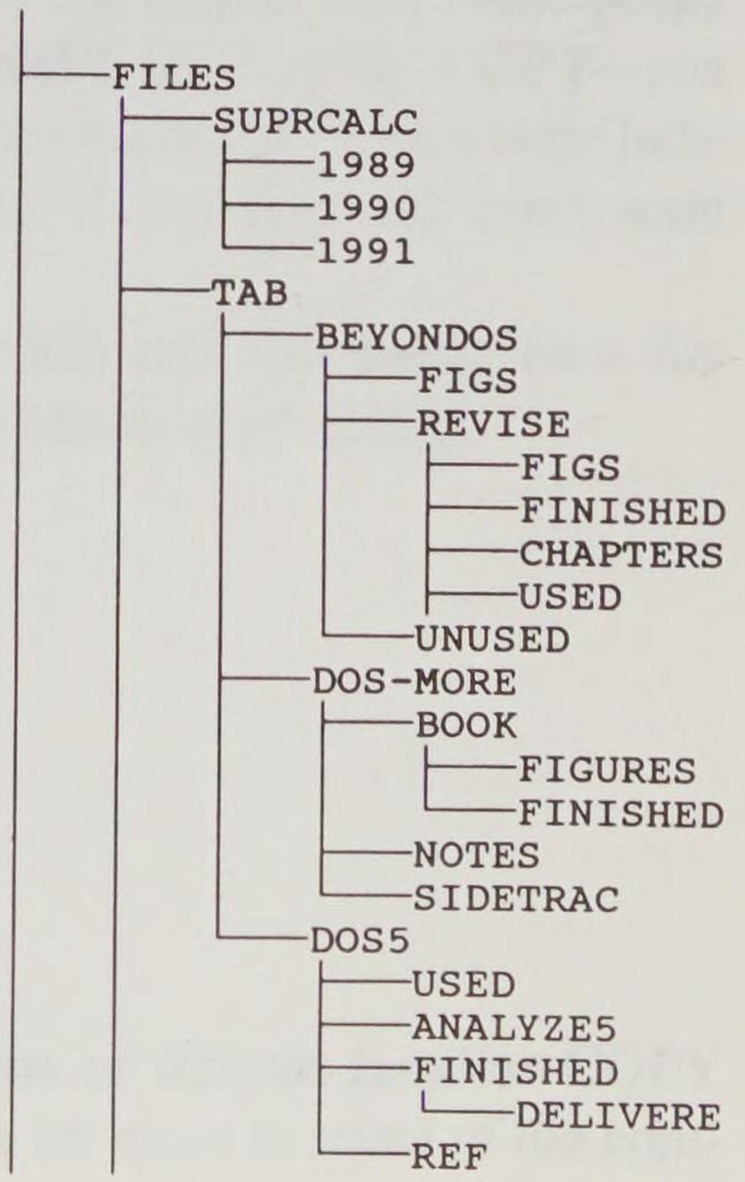
One peculiarity, mentioned elsewhere but worth repeating, is the MOVE function. This is totally different in operation from MOVE functions on some other shells. Many shells allow you to move files from one directory to another on the same disk by simply moving the directory listing, leaving the data sectors untouched—neat, clean, and fast.

The DOS shell doesn't do that, however. It moves things by copying them to new locations and then erasing the originals. Unlike 3rd-party shells that allow for the movement of files only between directories on the same disk, this one will transfer files between completely different disks—again, always erasing the originals when finished. That makes me nervous, although I was unable to make it lose any files that hadn't been properly copied for any reason—or at least that DOS didn't think had been copied correctly.

Around and around

You'll always come back to the directory tree itself. No matter how many clever shells you have or how slick the new DOS DIR command is in this release, it all comes down to how you organize your disks, and only *you* can manage that part of it.

Keep things grouped together logically. Not only is that imperative when it comes to finding things, but it greatly simplifies the problem of keeping backups not only current but complete. Here's the way I try to pattern the tree structure for all my files—all programs are in a separate DOS branch, as shown in Fig. 13-6.



13-6 Ultimately effective file management comes down to a matter of good disk organization, as shown in this tree display.

Not uncommonly, I'll have more than one project under the same general heading in progress at the same time—for instance, while working on this book I was also working on a second edition of an earlier book for TAB. So you'll notice that there's a whole subdirectory called REVISE—everything related specifically to that project. Going up just one level higher on the tree includes reference files from the first edition—and everything is still separate from any of my other TAB projects.

Yes, the use of these many levels probably slows DOS down, but I guarantee that it speeds me up a lot more than it slows DOS down. And the nice thing is that, no matter what sort of tangled web you have on your disk, it's never too late to sort things out and do it right. And now is the best time, with DOSSHELL's capability of moving things around. For all it might leave to be desired, at least it will do the job.

CHAPTER 14

Copycats

The copying of data from one place to another is one of the most critical functions of an operation, and while release 5.0 adds nothing to DOS's basic repertoire of tricks in this department—even the new DOSSHELL adds nothing really new—the subject warrants some discussion here because I think a lot of users fail to realize—or utilize—some of the better tools that DOS offers.

COPY—just plain COPY—is about the only copying command a lot of people ever use. Admittedly, that's probably the one you'll always use the most, but even plain old COPY does a lot more than a lot of users realize.

COPY, for example, supports a couple of rarely used switches—/a for ASCII and /b for binary (program files). Without going into a lot of unnecessary detail, just remember they're there the next time you have a problem with some pesky end-of-file markers, particularly when concatenating files with COPY—you know, stringing a bunch of little files together into one big lump. With a little judicious use of these two switches you can get rid of some markers you don't want and stick in EOFs that you do.

And did you know that you could change the time and date stamp on a file with COPY? Here's an old file with its original (or latest) time stamp:

```
SYNOPSIS.      1,840 .... 2-16-88 11:49:44 pm
```

And after running:

```
COPY [path]SYNOPSIS+,, [target]
```

you have:

```
SYNOPSIS.      1,840 .a.. 5-01-91 7:40:22 pm
```

Admittedly, these are lesser if not obscure functions of the old familiar COPY command, but I think they demonstrate that there's a lot more to some of the com-

mands you most likely take for granted than you realize. Just plain COPY does leave a lot to be desired—but then you're not stuck with just plain COPY, either.

Beyond the old familiar

In version 3, DOS introduced two powerful new utilities, XCOPY and REPLACE, both of which were and still are underutilized. Of the two, XCOPY is my favorite—in fact, though slower, I like it a lot better than BACKUP for making backups of my files because it keeps my files *as* files that I can access instantly, if needed, not as just a part of some big lump of files and data that require the same amount of disk space and must be restored before they can be used. The two main reasons I most often use XCOPY are:

- It recreates whatever directory structure the files were stored in on the source disk as it goes.
- It can pick and choose, moving only files that haven't been copied previously or that meet some other search criteria.

These are both options that must be set with switches when the command is entered, but the switches are logical and easily remembered—unlike those still in use with some commands. There are other options too. You can specify a cutoff date. Or you can even use it just like plain old COPY if you like.

In fact, in many cases XCOPY is significantly faster than just plain old COPY if you have a bunch of files to move. I used COPY to move a bunch of disk files to a RAM disk, and then did the same job with XCOPY for comparison. Times with XCOPY averaged about 85 seconds, and typically ran 25 to 30 seconds longer using just plain COPY.

The reason for the difference is fairly simple. COPY reads one file, then writes it, and then goes back to read another—one file at a time. XCOPY doesn't work that way. It sucks up all the files that it can hold in memory (or until it has them all) in one long uninterrupted read. Then it does one long uninterrupted write, repeating the process if need be.

There's a paradox in this, however, for unless you're monitoring the process with a stopwatch, XCOPY might *seem* to take forever. COPY, while it is going through its frantic read and write gyrations, displays a running account on screen. You know that it's copying files; you can almost *see* it happening. XCOPY, on the other hand, just sits there displaying *Reading source file(s)...* during the long, uninterrupted reads. It doesn't tell you which files it's reading, reporting only files it's writing as it writes them. The switches I find most useful with XCOPY are these:

/a	Copies files with the archive attribute set, but doesn't change the attribute.

/m	Copies files with the archive attribute set, and turns off the archive attribute.
/d:*date*	Copies files changed on or after the specified date.
/s	Copies directories and subdirectories except empty ones.

The archive bit

Attribute bits, particularly the archive bit, are something many users don't seem to understand, and it's confusing because, unless you use commands like XCOPY or REPLACE or BACKUP, they're something you never have to be involved with before. If you did, then it was most likely only in certain circumstances. There are four attribute bits in all:

- A Called the archive bit, is turned on when any new file is created or an old file has been modified.
- R If set, makes the file read-only so it can't be modified or overwritten.
- H Hides files from ordinary directory listings—but not with the new DOS 5.0 DIR /a:h switch.
- S Used for system files (such as IO.SYS and MSDOS.SYS), which in 5 for the first time can be seen in directory listings with the new DIR /a:s switch. (The two DOS system files, and many others, also show up in directory listings using the /a:h switch because they have that attribute set as well.)

To see which attributes are currently set, you can use the ATTRIB command without entering any parameters, and it will list the files in either the default directory or a directory you specify. For example, the command:

ATTRIB C:*.* and Enter

will produce the display shown in Fig. 14-1. All the main copying mechanisms check before overwriting any existing file of the same name, balking and issuing the following message if you try to overwrite one with the R attribute set:

Access denied – [*d:*][*path*][*filename*]
0 file(s) copied

```
      SH        C:\IO.SYS
      SH        C:\MSDOS.SYS
                C:\AUTOEXEC.BAT
   A            C:\CONFIG.SYS
   A            C:\COMMAND.COM
                C:\CURRENT.BAT
C:\MIRORSAV.FIL
        R       C:\OLD.C
```

14-1 In order to see which attributes are set, even with greatly enhanced 5.0 DIR command you still have to use the ATTRIB utility.

However, neither COPY nor any of the other copy mechanisms check the archive (A) attributes in their *default* copying modes, happily copying those with or without the A bit set with equal dispatch. Among the switches shown for XCOPY, however, are two specific to the A attribute: /a (which I included in the group above mainly for reference) and /m.

It's the /m option that I use most frequently, particularly when I'm working on some major project like this book. I use XCOPY to keep backups of all of my files on floppy disks, and /m assures that I have copies of all newly created files or any that I've updated since the last time. For example, Fig. 14-2 shows a partial ATTRIB listing from a fairly typical directory or working files. You have to be a little careful with this one, though.

```
A          DOSMOVES
           DOS_IS
A          DOS_IS     ADD
A          HEADER
           INTRO
           LOGIC
A          MASSTORE
           NEW_IN_5   CH
           READ_HEX
           SETVER
```

14-2 ATTRIB display showing files with the A (archive) bit set.

Four files in the figure have the A bit set, meaning they're either new or have been modified. I don't know exactly when, but it doesn't matter. I know—and XCOPY /m knows—they haven't been copied. At least not in any way that would affect the A bit. So if you specify:

XCOPY /m and Enter

it will just sit there for a moment saying:

Reading source file(s)...

and then report:

```
DOSMOVES
DOS_IS  ADD
HEADER
MASSTORE
     4  File(s) copied
```

In this case, this is exactly what you want to do, but let's check the attributes again:

```
DOSMOVES
DOS_IS
```

```
DOS_IS      ADD
HEADER
INTRO
LOGIC
MASSTORE
NEW_IN_5    CH
READ_HEX
SETVER
```

Now all the archive bits are switched off—the /m switch did it as it went. But suppose you want to run BACKUP now, this time just to add any new or modified files to your backup set. If you specify:

BACKUP /m and Enter

it will report:

WARNING! No files were found to back up

In other words, you can use this trick only once because, when the A bit is switched off, it's off. You have to be a little careful with this one. Yes, you can use the ATTRIB command to turn the A bits (or any of the other bits) on, either individually or in wild card groups or by any other sort criteria it might recognize, but that's a nuisance. The other possibility is you use the *copies* as the source for BACKUP, because copies, being newly created by definition, come into being with their archive bits turned on.

I use the /m switch deliberately as a part of my backup strategy—a strategy that's actually a two-stage backup. The first stage is an ongoing, virtually continuous, backup using a 3rd-party product called SITBACK, which automatically backs files up—including progressive changes—to a floppy that always stays in drive B:. Periodically, of course, that disk gets filled and SITBACK beeps at me. At that point I XCOPY all the files on the drive B: floppy to the backup disk using the /m switch. I can then check to be sure the archive bits have been switched off—confirming the fact that those files have been copied—before deleting the B: drive files to free the space for reuse.

There's a more positive way, however—and one that does switch off the archive bit. Use the /d:*date* switch instead. But for a one-shot, cheap-and-dirty, the /m switch does just fine.

Up that down tree again

One of the most powerful features of both XCOPY and REPLACE is the ability to back up wildcard-selected contents not just of a single directory but of that whole branch of the tree, keeping everything still sorted on the target disk the same as on the source.

The /s switch is used to include subdirectories with both XCOPY and REPLACE (and several other DOS commands) and can be applied selectively, but only to a point. Say, for example, that you have a directory tree starting at the root directory, which branches then in two primary directions, DOS and FILES, with each of those two branching farther—to a fourth directory level in some instances.

How well the /s switch works depends on how well your tree is organized, however. One of the primary criteria for creating a directory structure should be the ease of backing up the files you need backed up on an ongoing basis, and having all files hanging neatly from a FILES branch of the tree immediately segregates FILE files from program files, so even the command:

```
XCOPY \FILES\*.* target /s/m
```

would immediately exclude everything on the DOS side of the tree and copy only new or modified files on the FILES directory.

A lot of times that isn't good enough, though. I usually have several projects going at the same time, and each one has its own branch off of the main FILES directory: DOS5 for this book, BEYONDOS, for an earlier book (*MS-DOS Beyond 640K*), and so forth, so I can narrow the backup criteria still farther by doing something like:

```
XCOPY \FILES\BEYONDOS\*.* target /s/m
```

But that still isn't really what I need to back up on a daily basis. While working on a new edition, it's the REVISE subdirectory and everything below that that I really want, and that's easily done with:

```
XCOPY \FILES\BEYONDOS\REVISE\*.* target /s/m
```

Now I'm getting somewhere. This will recreate the specified mini-tree, if it doesn't already exist on the target disk, and copy any new or modified files it finds in any of those slots onto the target disk.

What you can't do with this command (or any simple modification) is *exclude* directories that aren't important. I have one called USED that has wreckage of some old files I've totally rewritten—this is stuff I'll probably never need—but I'd just as soon not throw it out until the project's finished. So it really isn't worth backing up. However, there's no direct way of doing this with DOS. If you wanted to work from a batch file, you could do something like:

```
ATTRIB \FILES\BEYONDOS\REVISE\USED\*.* -A
XCOPY \FILES\BEYONDOS\REVISE\*.* target /s/m
```

There are other tricks you could use to, but personally I think a lot of the cute little things you *can* do are more nuisance than they're worth. The /s switch is a real time saver if you have your directories laid out, though. This one and XCOPY are a real dynamic duo.

Replacing what was never there

REPLACE is another powerful disk and file management utility. Like XCOPY, it can handle subdirectories with an /s switch, the most significant difference being that its main purpose is to *update* files that already exist, skipping over any that don't. REPLACE also has an /a switch for adding files that didn't previously exist on the target disk. This switch can't be used in combination with other switches supported by REPLACE, however. This one is great for keeping current backups of electronic checkbook registers, PIMs, and other databases where your only interest is a certain group of files whose filenames never change—which, in most cases, they rarely do.

And because it has to update only files with names that already exist on the target disk, this one lends itself especially well to batch file use. You could, for example, keep a backup disk in drive B: and include a REPLACE [*path*]*.* /s statement in your AUTOEXEC.BAT to do an automatic update of those crucial files every time you boot your system if you like.

This one also differs from the way XCOPY works, in that it does *not* support an /m switch. It does even better, however, with a /u (update) switch, which simply checks the timestamp, leaving the archive bit the way it was (either on or off) when it does its job. It's too bad XCOPY doesn't do that, but it doesn't. Not yet, anyway.

The REPLACE /a switch will put files on the target disk that weren't there previously, after which subsequent REPLACE commands will try to match those names against whatever files otherwise meet the match criteria. But then you can probably do that just as easily, maybe even easier, manually with just COPY. To me, this one's a nuisance, and what matters to REPLACE isn't how filenames got there, only that they're there. So make it easy on yourself.

Safety in numbers

Disk problems tend to come in one of two sizes—ordinary run-of-the-mill disasters that can wipe out days or weeks of work in an instant, or the really bad kind. With the latter kind, you'll typically turn on your machine one day, all ready to go to work, and, with your screen still mostly black as is fitting for such occasions, DOS reports something like:

```
Boot disk failure, sector not found
```

or more precisely:

```
Error at 0000:0E13
```

or perhaps just:

```
General failure
```

or something equally cryptic—and terrifying. Or your applications software—Lotus or dBASE or WordStar—might just not load. Or you get garbage on your screen. Or whatever. You'll know. Sooner or later . . . you're going to know. Of course, all of this points up how important it is to take precautions now!

Given the powerful new data protection tools provided by DOS 5.0—MIRROR, UNFORMAT, and UNDELETE, there are, in fact, few disk disasters that are truly unrecoverable. Just as long as you don't panic—all too often it's sheer panic rather than the event itself that ultimately does the greatest damage.

All this high-tech doesn't mean that you can be complacent about the use of more conventional data protection methods. There's no single best tool—or for that matter any single tool that really does it all. Not to my satisfaction, anyway.

Failsafe disk and file protection can be accomplished only through a comprehensive strategy that brings a variety of tools to bear on the problem, starting with a plain old-fashioned backup of your precious files and data. The need for that hasn't diminished. If anything, the need grows even greater as the number of important files you've accumulated keeps getting bigger. The more you have to lose, the more you can afford to leave no stone unturned in this regard.

DOS, going clear back to version 2, before most users even had a hard disk, has given you the means of backing up your files—BACKUP and RESTORE. These two utilities are still a part of DOS 5.0 and, as such, still fill the same niche in a backup strategy: BACKUP still backs up, and RESTORE still extracts selected (or all) files from your backup file and puts them back in place.

DOS's BACKUP utility isn't that different in its operation from some of its pricey competition. What has always been most lacking is a friendly user interface. That, unfortunately, hasn't changed in this release. The only thing that 5.0 really adds in this regard is DOSKEY macros as a means of automating backups so you don't have to keep the switches and the syntax all in mind.

By and large I find that my backup needs don't really change that much. Not often, anyway—and again, with proper directory management, I find that changes brought about by adding new directories can be picked up almost if not completely automatically.

If you do it right, you'll have to sit down only once to figure out just what you need backed up and how you want it done. Then just stick it in a macro (or a batch file) and forget the details. There's nothing glamorous about the DOS BACKUP utility. It does the job, though, and the price is certainly right—no extra charge when you buy DOS.

Actually, BACKUP in itself is probably a better utility than many users give it credit for. At least one vendor of tape backup drives has actually used the DOS BACKUP and RESTORE utilities to do the actual work, masking that fact behind a friendly menu-driven interface. It's the DOS interface—or lack of one—that puts most users off, although there are some other factors to consider.

Less-than-perfect security

One of the things I don't like about BACKUP—or any of the numerous comparable 3rd-party backup utilities I've tried—is that backed-up files aren't saved as individual free-standing files. You can't simply go to a backup disk, find a file you might have lost (or that might have been damaged), call it up, and use it. BACKUP concatenates files—strings them together into one big lump (often even spanning several floppy disks), and creates an index as it goes along. Figure 14-3, for example, shows the directory listings from the first two floppies of one backup.

```
 Volume in drive A is BACKUP  001
 Volume Serial Number is 105A-17FC
 Directory of A:\

BACKUP   001    361472 05-01-91   2:22p
CONTROL  001       889 05-01-91   2:22p
        2 file(s)     362361 bytes
                        0 bytes free

 Volume in drive A is BACKUP  002
 Volume Serial Number is 2C70-11E8
 Directory of A:\

BACKUP   002    360448 05-01-91   2:25p
CONTROL  002      1809 05-01-91   2:25p
        2 file(s)     362257 bytes
                        0 bytes free
```

14-3 Directories of two backup disks prepared with DOS BACKUP. There's no way to tell which disk contains any specific files that might be needed except by running RESTORE on each disk.

In this case, I was backing up just one rather large directory of just over 4 megabytes of files in two dozen subdirectories. If I'm looking for only one file, where do I look? Not in the index file (CONTROL.*nnn*) because it's not legible (it's in machine code). So I have to send DOS looking, inserting the backup disks one-by-one as requested, until the file is located. With the command:

```
RESTORE A: B:\BEYONDOS\CRYSBALL.C16
```

you'll receive the following screen display:

```
Insert backup diskette 01 in drive A:
Press any key to continue...

Insert restore target in drive B:
Press any key to continue...
```

```
*** Files were backed up 05-01-1991 ***
*** Restoring files from drive A: ***
Diskette: 01
Insert backup diskette 02 in drive A:
Press any key to continue...
*** Restoring files from drive A: ***
Diskette: 02
```

and so forth. But wait a minute. After all of this, DOS has skipped right over the file I was looking for (it was on DISK 01) due to some error in entering the command—something too minor to set off the alarm bells but serious enough to keep DOS from finding the file I was looking for. You can run both BACKUP and RESTORE through the 5.0 shell, but unfortunately the shell does nothing that makes the process any easier.

One of the other problems with RESTORE is that it will restore files only to the same directory as they were located in originally. Normally, this is probably not a problem, but if I have a file I know is damaged, I would rather check whatever backup I might have to be sure it's valid before copying over the damaged original. RESTORE does allow you to restore to a different drive—say to a floppy rather than back onto your hard disk, but only as long as you let it duplicate the original directory structure.

So while BACKUP really isn't a bad utility in itself, I don't like RESTORE. And because your backups aren't much good without RESTORE . . . As a purely practical matter, If you're like me I think you'll like XCOPY a whole lot better for most backup jobs.

CHAPTER 15

Close encounters and disasters

Aside from memory management, the most important changes in version 5.0 are in the area of disk management, much of it based on technology licensed from Central Point Software (PC Tools) but modified to some degree by Microsoft. At the "Close Encounters" end of the spectrum, this new release adds UNDELETE, a new utility that, used as a stand-alone, brings to DOS an ability to recover deleted files found previously only in 3rd-party packages, the Norton or Paul Mace Utilities, PC Tools, etc.

That's not the story—not the real one, anyway—because it's a whole new ball game starting with a nondestructive FORMAT, allowing even for the total reconstruction of a disk—hard disk or floppy—that has been accidentally or just prematurely reformatted with UNFORMAT, another new addition.

And even that, while nice and certainly a welcome improvement, is only the tip of the iceberg, because, with this release, DOS even has the power to recover your hard disk after all but the most catastrophic crashes with the help of MIR ROR, another new utility.

MIRROR is peculiar in that sometimes when it's finished doing what was asked of it, it terminates and goes away. Other times it's a TSR. In one mode—when it's a TSR—it keeps track of any files you've erased to make recovery easier than Norton or most 3rd-party unerase utilities (except PC Tools). MIRROR even plays a role whenever you format a disk.

But mirroring the FAT and other critical disk data is its most important role, because few disk "crashes" are really that. With MIRROR, DOS now has the

power to fix almost all but the most serious (mostly mechanical or electrical) hard disk problems.

Fortunately, most data recovery problems can be solved with simpler solutions—or if not really simpler, at least more conventional. And DOS 5.0 can handle those as well.

The simple "OOOPS!"

It happens to everyone, expert and novice alike—one too many wildcards when you're cleaning out old files, or you find you're in the wrong directory. Too late. They're gone. Not really, however, because DOS doesn't really *erase* the old files. Except for reformatting floppy disks, which is another story, all DOS does when you erase a file (ERASE or DEL, it's the same either way) is simply drop it from the directory listing. Until DOS has to use the space again, the file itself is still intact right where you left it. In fact, all DOS did was write a funny squiggle over the first character in the root directory entry. The squiggle told DOS not to list the file in any call for a directory listing, so for all intents and purposes the file was gone.

Because of this, the recovery of erased files is hardly new. What's new are DOS utilities to help you do it—and not just at some rudimentary level. 5.0 DOS supports the recovery of accidentally erased files at not just one but two different levels with the new UNDELETE utility. At the most basic level, what DOS offers is comparable to what you'd expect with either the Norton or Mace utilities.

Tracking and recovering erased files

DOS 5's UNDELETE goes one step further than the more typical 3rd-party unerase utilities—at least it can, given a little help from MIRROR. Mirror—which has its origins in PC Tools—is more than just a program. It's a mechanism and, as such, appears in several guises and performs several different though related functions.

In one of its several manifestations, loaded as a TSR, a part of MIRROR .COM keeps special track of every file that you've erased in a file called PCTRACKR.DEL. To do this, load MIRROR by simply entering the following command specifying one or more drives to track at the DOS prompt (or from a batch file):

```
MIRROR /td [/td]
```

Anyone familiar with most of the 3rd-party unerase utilities will spot the difference rather quickly. If, for example, you issue the command UNDELETE B:, you'll see what's shown in Fig. 15-1. Yes, DOS 5.0, like all prior versions, still throws away the first character of the filenames of erased files, as shown in the glimpse at part of the contents of the root directory in Fig. 15-2.

```
Directory: B:\
File Specifications: *.*

    Deletion-tracking file contains    4 deleted files.
    Of those,     4 files have all clusters available,
                  0 files have some clusters available,
                  0 files have no clusters available.

    MS-DOS directory contains    6 deleted files.
    Of those,     4 files may be recovered.

Using the deletion-tracking file.

     NETWORK            863  1-29-91 10:00p  ...A  Deleted:  3-19-91  5:27p
All of the clusters for this file are available. Undelete (Y/N)?y

File successfully undeleted.

     UPGRADE           1526  2-01-91  1:18p  ...A  Deleted:  3-19-91  5:26p
All of the clusters for this file are available. Undelete (Y/N)?
```

15-1 If MIRROR with the delete tracking feature (TSR) is used, the slick new DOS UNDELETE utility even recovers complete filenames automatically.

```
NETWORK          863   1-29-91   10:00 pm    404    Arc
DOS_IS         16208   11-18-90   8:27 pm     98    Arc
σPGRADE         1526   2-01-91    1:18 pm    114    Arc
```

15-2 A look down at the root directory level reveals that the first character of the filename of an erased file has been changed.

And in fact, when MIRROR has not been run as shown above, UNDELETE works just like almost any other unerase utility that works only from the remaining entry data in the root directory, prompting for the lost first character in every filename you want recovered, as shown in Fig. 15-3.

Otherwise, when used in conjunction with MIRROR, UNDELETE has several switches that allow you to select between recovering erased files listed in the PCTRACKR.DEL file or any recoverable entries from the DOS directory. It seems wise to be consistent, though, either running MIRROR with the /t*d* (tracker) switch at all times on a given drive—as from the AUTOEXEC.BAT—or not at all.

The amount of disk space used up by the PCTRACKR.DEL file is minimal, depending on disk size and the number of entries allowed for. Table 15-1 gives approximate file sizes, using the default number of entries. Starting with a freshly formatted disk, you could increase any of these numbers significantly in most cases, should you want to use the /t*drive*[-*entries*] switch with MIRROR and create a larger PCTRACKR.DEL. These numbers should be adequate for most purposes, however.

UNDELETE can handle subdirectories as easily as the root directory too, as long as the subdirectory itself hasn't been removed (RMDIR or RD). You do have to specify which subdirectory you want processed on the command line, however,

```
Directory: B:\
File Specifications: *.*

    Deletion-tracking file not found.

    MS-DOS directory contains    2 deleted files.
    Of those,    1 files may be recovered.

Using the MS-DOS directory.

    ?HELL                4573 11-26-90  2:03p  ...A  Undelete (Y/N)?y
    Please type the first character for ?HELL    .   : s

File successfully undeleted.

    ** ?ETWORK                863  1-29-91 10:00p  ...A

Starting cluster is unavailable. This file cannot be recovered
with the UNDELETE command. Press any key to continue.
```

15-3 Here, the delete tracking feature of MIRROR hasn't been used and first letters of filenames are lost and must be filled in.

Table 15-1 The default maximum number of entries and approximate amount of space required by MIRROR files for various disks or disk partitions.

Disk size	Number of entries	File size
360K	25	5K
720K	50	9K
1.2Mb	75	14K
1.44Mb	75	15K
20Mb	101	18K
32Mb	202	36K
Larger	303	55K

or it will presume you want the default and quite likely scare the hell out of you by telling you there's nothing to unerase, even though the PCTRACKR.DEL file itself is always on the root directory. You can't specify a particular filename within a directory from the command line, but the syntax is quite simple and direct:

UNDELETE *d:*[*subdirectory*]

Don't forget, though, that if you optimize (unfragment) your disk, any unrecovered files are history, so, as in all things, there are choices to make.

As a practical matter, it seems that if you accidentally dump a file, you usually know about it rather quickly—and at that point the odds are pretty good with UNDELETE. This one is good, fast, and easy to use. It's definitely a welcome

addition to DOS—with or without a PCTRACKR.DEL file. There are, however, some practical considerations that come into play and can severely limit the ability of MIRROR (or any other file or data recovery scheme). And these need to be understood, lest you lull yourself into a sense of false security.

As long as DOS doesn't have to reuse any of the disk space that has remained idle since the disk was last formatted, every single file you've erased is potentially recoverable. As your hard disk starts to fill and there is less and less abandoned or erased space for DOS to draw on, when you write new files the length of time an unused space remains unused again gets shorter and shorter.

Carried to the extreme, you've probably had the occasion, especially with floppies, to fill a disk to a point where DOS is unable to copy even one more file to it unless you go in and erase another file of comparable size. If you erase that one file and then copy a new file that barely fits the space you've emptied, that old file you just erased is gone beyond recovery—once DOS copies over it, it's gone.

A kinder, gentler format

As many of you already know, DOS never did do destructive formats when it came to hard disks. Only to floppies. Capitalizing on this, Paul Mace, Peter Norton, and several other 3rd-party software developers through the years have marketed a variety of increasingly sophisticated hard disk recovery programs. Along with these, because it isn't really necessary to destructively format floppies either, a number of DOS-compatible nondestructive 3rd-party format programs have emerged, as well, with PC Tools introducing one of the later but possibly better known of these.

It's vitally important that anyone familiar with the nondestructive formatting techniques used in the format utility supplied with the Norton Utilities, as well as other 3rd-party format utilities that offer unformat capability, understand the differences in the way DOS provides for the recovery of files from disks that have been reformatted. This is critical because the files and data on any disks reformatted by DOS 5's new nondestructive format are recoverable only by DOS version 5.0. They are, in fact, apparently nonrecoverable even using the PC TOOLS utilities from which they were developed.

While the new DOS formatter doesn't corrupt the actual data, no files on a DOS-formatted disk can be recovered by the Norton Utilities (or similar recovery utilities). This isn't presumption; it's fact. And it doesn't stem from a determined effort on the part of Microsoft to make life difficult, but rather from their having taken an entirely different approach to the problem.

If you use Norton's SF (Safe Format) formatting utility, for instance, not only does it not overwrite any of the data sectors (default mode), but it also doesn't overwrite the FAT and, as shown in a 720K disk in Fig. 15-4, the root directory entries. Several nonprinting characters are missing from the display as reproduced.

```
 Sector 7 in root directory                                    Offset 0, hex 0
E54E5349 20202020 53595320 00000000 00000000 0000C02C σNSI    SYS .........└,
5A150200 95220000 E5505045 4E442020 45584520 00000000 Z. .."..σPPEND  EXE....
00000000 0000C02C 5A150B00 66290000 E5535349 474E2020 ......└,Z*δ.f)..σSSIGN
```

15-4 A typical 3rd-party nondestructive formatter—Norton's SF in this case—preserves all important file data in the root directory, treating all disk contents as erased files.

These entries simply show the files as having been erased—the first character of the filename is removed and replaced, but all other data, including the locations of all the scattered bits and pieces for all of the files, are still on the disk. That data is untouched even after reformatting by the Norton formatter. Until overwritten by new files, any and all files existing on the disk before reformatting can then be normally recovered by simply *unerasing* them.

The new DOS FORMAT doesn't work that way. It *does* overwrite the FAT and root directory. As shown in Fig. 15-5 (the same disk and same files in Fig. 15-6 copied back to it, but reformatted by the new nondestructive DOS FOR MAT), it wipes out the critical root directory information completely.

```
 Sector 7 in root directory                                    Offset 0, hex 0
00000000 00000000 00000000 00000000 00000000 00000000 ........................
00000000 00000000 00000000 00000000 00000000 00000000 ........................
00000000 00000000 00000000 00000000 00000000 00000000 ........................
00000000 00000000 00000000 00000000 00000000 00000000 ........................
```

15-5 The root directory of disk reformatted with new DOS 5.0 nondestructive formatter. Although all data is still intact, all root directory and FAT entries have been completely zeroed out, making recovery of any files impossible, except with the new DOS UNDELETE or UNFORMAT utilities.

```
MIRORSAV.FIL        41 rash   3-19-91   5:25:24 pm
MIRROR  .FIL    50,688 ra..   3-19-91   5:25:26 pm
```

15-6 If you really want to look good, run MIRROR on your hard disk regularly. Here, in only about 50K, MIRROR has preserved the entire root and FAT data for a 33Mb hard disk.

Clearly, there's nothing left—no road map, so to speak, to follow, which is why you can't use the Norton (or similar) utilities to recover any files at that point. The same is true even with the new /q (quick) DOS FORMAT option. Data, yes, but only on a sector-by-sector, dig-it-out-the-hard-way basis, and the same applying to both hard and floppy disks now with version 5.

Obviously, for files to survive reformatting, the FAT and root directory data has to be retained somewhere, and DOS does save it, but to a different location on the disk. It saves this information to erased or unused filespace where it doesn't overwrite any valid recoverable file—but in a way that only the DOS UNFOR MAT utility can find to restore the FAT and root directory entries from it. It's so well hidden that FORMAT reports the entire formatted disk space as being available for use (less any bad sectors, of course).

This requires that there be a certain minimum amount of unused or erased

filespace available before reformatting, for the formatter to copy that data back to without overwriting any of valid files and data you might want to recover. If it did that, it would be defeating the purpose. So if you have a floppy disk that's very nearly filled and you issue the following command:

```
FORMAT b:/f:720
```

You'll see the following information:

```
Insert new diskette for drive B:
and press ENTER when ready...

Checking existing disk format.
Saving UNFORMAT information.

Drive B error. Insufficient space for the MIRROR image file.
There was an error creating the format recovery file.
This disk cannot be unformatted.

Proceed with Format (Y/N)?
```

At this point, you have the option of aborting the format so you can erase an old file or two—something you're sure you couldn't possibly need to recover—to free the space DOS needs to store the data it needs. Or you can proceed and the format will be nonreversible—which, in many cases, particularly in the case of a floppy disk, is likely not a problem.

The *MIRROR image file* that the message warns about is another story, and is the key to DOS's new disk management, disk recovery, and data protection scheme.

Mirror, mirror, on the wall . . .

Surely the most significant enhancement to DOS's disk management capabilities to date is this MIRROR thing—at times a special program, sometimes a TSR, at other times only a lesser function of a greater whole. As a program, MIRROR is chameleon-like, changing its whole *modus operandi* to suit a variety of rather diverse tasks. And then there are the times you haven't run it—that the MIRROR program isn't even there perhaps, and yet the MIRROR image files show up anyway as if it was . . .

Confused? You should be—and one of the things that complicates getting a handle on exactly what DOS 5.0 is doing that enables you to recover from an accidental reformatting is the way DOS now creates, or at least tries to create, a MIRROR image file whenever you reformat a disk.

Confusing the issue even more, though, is the fact that DOS 5.0 also has a separate new utility called MIRROR.COM that can be used to create special tracking files for erased files, a record of the partition tables on a hard disk, a record of the FAT and root directory entries, etc. That's what MIRROR.COM

does—but you can locate these files. They have the attribute bytes set to hide them from a normal listing, but they're there, as shown in Fig. 15-6.

But when you format or reformat a disk with the new DOS 5.0 FORMAT .COM, you also wind up with something DOS refers to as a *MIRROR image file*—if DOS has anything to say about it. In fact, you wind up with a MIRROR image file, which is really just data rather than a conventionally structured file, even if MIRROR.COM is nowhere to be found.

This is the key, then, this MIRROR image file. One key anyway, because without that MIRROR image file, there's no tomorrow—and yet there seems to be two kinds of MIRROR image files. And here Microsoft is rather stingy with their documentation. Not only stingy, but ambiguous—I wound up going back to PC Tools documentation at one point and that helped. However, as finally implemented in DOS, it's significantly different in detail from its progenitor, so it still took some deeper digging before I was completely satisfied.

To check it out—and ultimately to illustrate the point—I erased MIRROR .COM from my hard disk, then proceeded to reformat a disk. A quick directory check after formatting revealed no files, mirror or otherwise in the directory—not even a hidden file. The MIRROR file was there when I ran UNFORMAT on drive B:, though:

```
Insert disk to rebuild in drive B:
and press ENTER when ready.

Restores the system area of your disk by using the image file
created by the MIRROR command.

WARNING !!              WARNING !!

This command should be used only to recover from the inadvertent
use of the FORMAT command or the RECOVER command. Any other use of
the UNFORMAT command may cause you to lose data! Files modified
since the MIRROR image file was created may be lost.

Searching disk for MIRROR image.

The last time the MIRROR or FORMAT command was used was at 18:41
on 03-18-91.

The MIRROR image file has been validated.

Are you sure you want to update the system area of your drive
B (Y/N)? y

The system area of drive B has been rebuilt.

You may need to restart the system.
```

Now you see it, now you don't—and now it's back again. In fact, after using UNFORMAT I ran UNDELETE and even recovered some (but not all) of the files that had been erased before reformatting. And something else. After running

UNFORMAT, there's evidence of not just one but two erased files, MIRORSAV .FIL and MIRROR.FIL, along with the other entries in the root directory (that funny ASCII character replacing the first letter of the filenames indicates those files had been erased), as shown in Fig. 15-7.

```
INSTALL  SOP      53234  11-23-90  10:37 am     33    Arc
README   DOC       7573  11-23-90   1:40 pm     85    Arc
σIRORSAV FIL        41   0-00-80  12:00 am    714    Arc R/O Sys Hid
SCHECK   EXE      34872  11-21-90   5:54 pm     93    Arc
σIRROR   FIL       6656  12-03-90   2:34 pm     92    Arc R/O
SCREXEC  EXE      59199  11-23-90  10:54 am    130    Arc
```

15-7 After running UNFORMAT to restore root directory and FAT with data from the MIRROR file, even erased files still appear as erased files down at the root directory level.

Although there was no trace of them to be found on the formatted disk, after *unformatting* they show up in the root directory—with the same filenames as if you had run MIRROR.COM—but as erased files now, having done their job.

The ultimate test

Of course, while it's nice—and safe—to try out things like UNFORMAT on floppy disks as in the example on page 160, a hard disk is a different matter. Obviously the principle is the same, so if it works with floppy disks it *should* work with hard disks too. Unfortunately, there's only one way to be sure—you don't have to be crazy in this business, but it does help . . .

So I ran MIRROR to save the FAT and root directory on my hard disk. And then I deliberately unfragmented the disk—which took a while because the hard disk on the machine I tried this on hadn't been optimized for quite a while. Of course, at that point the MIRROR image file created by MIRROR.COM was invalid and it wasn't updated to reflect the changes made during unfragmentation before reformatting the disk with DOS 5.0 FORMAT utility in its default mode—maybe you do have to be a little crazy after all. Anyway, I cold booted the machine from a 5.0 floppy and unformatted it with the command UNFORMAT C:, and got the following results:

```
Restores the system area of your disk by using the image file
created by the MIRROR command.

WARNING !!          WARNING !!

This command should be used only to recover from the inadvertent
use of the FORMAT command or the RECOVER command. Any other use of
the UNFORMAT command may cause you to lose data! Files modified
since the MIRROR image file was created may be lost.

Searching disk for MIRROR image.
```

```
The last time the MIRROR or FORMAT command was used was at 16:31
on 03-22-91.
The prior time the MIRROR or FORMAT command was used was at 15:47
on 03-22-91.

If you wish to use the last file as indicated above, press L.
If you wish to use the prior file as indicated above, press P.
Press ESC to cancel UNFORMAT.
L

    The MIRROR image file has been validated.

    Are you sure you want to update the system area of your drive
    B (Y/N)? y

    The system area of drive B has been rebuilt.

    You may need to restart the system.
```

As you can see, if you've run MIRROR you have a choice—which MIRROR image file to use. Obviously, in this case only the one created by FORMAT.COM, the later one, was valid. It took about 20 seconds of busy hard disk activity—you could hear that little sucker just working up a storm, but it brought it back, the whole works. It was all there—776 files in 40 directories—some of them four levels deep.

What this demonstrates conclusively is that the new DOS formatter carries out its role of nondestructive formatting in a way that allows recovery without any help from MIRROR.COM; by itself it can and does create and write-back image files that are at least functionally identical to those created by MIRROR.COM, and from which UNFORMAT can, provided you haven't overwritten something in the meantime, restore your entire disk to its preformatted condition.

Not really what it seems

UNFORMAT is a strange program and the name is actually ambiguous when you come right down to it. It can, with any luck it seems, restore almost any disk that's had its FAT or root directory (or both) damaged—it doesn't have to have been reformatted, just corrupted. Provided you've run MIRROR.COM, that is, and I specify the .COM part to differentiate it from the mirroring function of the new formatter.

Take RECOVER, for example, one of DOS's all-time deadliest commands. It's designed to help recover damaged disks, where saving even bits and pieces of the data is worthwhile, but if you forget to specify a filename (RECOVER won't accept wild cards), it will turn your whole directory into virtually nonrecoverable mush, like what's shown in Fig. 15-8.

```
 Volume in drive B has no label
 Volume Serial Number is 1E74-17FC
 Directory of B:\

FILE0001 REC      33792 03-21-91    6:02p
FILE0002 REC      37888 03-21-91    6:02p
FILE0003 REC       1024 03-21-91    6:02p
FILE0004 REC       2048 03-21-91    6:02p
FILE0005 REC       1024 03-21-91    6:02p
FILE0006 REC       1024 03-21-91    6:02p
FILE0007 REC      19456 03-21-91    6:02p
FILE0008 REC      48128 03-21-91    6:02p
```

15-8 The entire directory of a disk turned to mush by careless use of the RECOVER command.

And it will continue to mush your disk until it uses up all the spaces in the root directory, wiping out all trace of any other files you might have had. At least as far as finding them is concerned; the data is still there, even on a floppy with the new DOS formatter. But if you've had the foresight to run MIRROR lately, so is a copy of the FAT and root directory—at least as of the last time that you ran it. With that MIRROR image file, all you have to do is UNFORMAT the disk. So it wasn't reformatted you say?

It doesn't matter, and the reason that it doesn't matter is that unformatting really isn't what UNFORMAT does. It's a FAT-fixer—FAT and root directory, even partition table fixer. That's what it really does. Unformatting just happens to be one very serious case of something down there needing to be fixed.

And if the FAT and root directory just come up garbage some day, if you've used MIRROR faithfully, barring some internal mechanical or electronic failure of the drive itself, with any luck UNFORMAT can put you back in business in just seconds. This feature wasn't documented at the time this was written, apparently through some oversight, but this is a proper and fully supported feature of the UNFORMAT command.

Note: you don't have to reformat your disk to run UNFORMAT. Again, the formatting process has nothing to do with what the UNFORMAT program does, other than assure that there is a current MIRROR image file. But if the drive seems to have failed, the *last* thing you want to do is save a copy of the damaged FAT, etc. It isn't the reformatting, but rather having a very recent MIRROR image file that matters.

More than the proverbial ounce

Obviously, you should never rely too heavily on being able to resurrect lost files, but it's nice to know it can be done and that now DOS gives us the tools you need

to do it—and do it with MS-DOS utilities, whether or not you've taken the trouble to have MIRROR keep track of all erasures—the program MIRROR.COM this time, not the "phantom" MIRROR that does secret, hidden image files whenever you format or reformat a disk with 5's new formatter.

While I probably erase as many files by accident as anyone, I'm not sure that having a tracker file is that important. The necessity would seem even more nebulous in the case of floppy drives, but it's a matter of choice and if you want it, it's there.

And realistically, the likelihood of wanting to recover from an accidental reformat should be virtually zero with floppies and, with any reasonable care, minuscule even with hard disk because—unlike DOS 2.x, which seemed to take delight in mushing your hard disk at the least provocation or oversight in entering the FORMAT command—the new formatter won't reformat your hard disk under any conditions without at least pausing and checking to be sure that's what you really want to do. If you issue the command FORMAT C:, you'll receive the following warning:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE C: WILL BE LOST!
Proceed with Format (Y/N)?
```

The question then comes down to "How much protection do you really need?"

The real value of MIRROR.COM is in preserving a copy of your FAT and root directory against the day your hard disk crashes. Surprisingly, this is something barely even mentioned in the Microsoft documentation to date. But this insurance by itself is worth more than the price of admission. If you're not already running MIRROR or something like it, you're really flirting with disaster—and you'll never miss the few seconds it takes to run MIRROR from your AUTO EXEC.BAT to keep the image file updated. Right after the line that runs CHKDSK to be sure the data MIRROR is going to save is good.

Saving system information

No matter what other form of backup you use, there are two areas on your hard disk that deserve special attention—and which ordinary backups overlook completely. These are the FAT (*File Allocation Table*) and the root directory, which are typically under 40K for a 20-megabyte hard disk (or partition).

There is, of course, no substitute for a comprehensive backup of everything on your hard disk, and none of the features of MIRROR should be construed in any way as eliminating that need. What a backup of the FAT and root directory buys you is a chance, in many cases, to recover a hard disk that has crashed catastrophically and might not allow access to files.

This is something I've been advocating—and doing—since long before DOS

provided any means for doing it, so I know the scheme is sound. True, simply backing up the FAT and root directories isn't a panacea. But in the many cases when it does work, it can save more than your day.

And there are some practicalities to consider here beyond the obvious—practicalities based on a natural tendency most of us seem to have to put off until tomorrow or the next day the backup that we should have done today. Or last week maybe.

MIRROR backed up the FAT and root directory of a 33Mb partition on my 16Mhz machine in just under 5.16 seconds—50,688 bytes—and we can all afford five seconds of our precious time. If you put MIRROR in your AUTOEXEC.BAT to do it automatically, you'll never even miss it and you'll *always* have a backup that's up to date.

Note, though, that if you unfragment your hard disk periodically, you must update your MIRROR FAT and root directory backup and erase any MIRROR .BAK you have on your disk if you haven't used the /1 switch. (While DOS still doesn't offer a utility that does it, I highly recommend it, not only because it significantly increases the chance of recovering lost data if all else fails, but because it also makes your system run faster in the bargain.) This is critical because unfragmenting actually moves files to new locations on the disk. The process of unfragmenting involves:

1. Finding a chunk of contiguous disk space big enough to hold the entire file.
2. Collecting the scattered bits and pieces.
3. Copying them into it sequentially.
4. Erasing the fragmented bits and pieces to free the space they occupied.

Having finished with one file, the unfragmenter moves on to unfragment the next and then the next, etc. By the time it's finished, lots of things have been relocated. If you fail to run MIRROR again to update your files and then send it looking for them in the old locations where DOS might have already put parts of other files . . . you don't even want to thing about the consequences.

However, even if you don't unfragment your hard disk you should update your MIRROR file on a regular basis—ideally every time you boot your system to be sure your MIRROR image file is up to date because, for obvious reasons, it can't help in the recovery of any files created since the last time that you ran it.

Use it because most often, even if it seems your whole hard disk is gone, it's just a problem—usually correctable—in these critical few sectors that tell DOS how your disk is organized and where things are. And with MIRROR, DOS now gives you a second chance; a chance of full recovery, probably in less than a minute—once you've regained your composure or whatever else you might have lost when suddenly your computer became stupid.

There's one more piece in the puzzle that shouldn't be overlooked, and that's another function of MIRROR.COM—again, MIRROR.COM and UNFORMAT .COM together. This is MIRROR run with the /partn switch to save a backup copy of the partition table of your hard disk, too.

Saving hard disk partition data

The hard disk partition is really nothing but a small file. But it's an extremely critical small file. As one of its several functions, MIRROR can save that data to a file called PARTNSAV.FIL. It's a short file, often as little as only 2048 bytes in length. Yet this tiny file is so important that should the original become corrupted on your hard disk you just plain don't have a hard disk.

To understand the importance of the partition table, you need to understand a little of its purpose. When you buy a new hard disk, unless the vendor has prepartitioned it for you, all you're really buying is a totally blank media of indeterminate size. Partitioning magnetically marks a section of that media and tells DOS (or whatever operating system you're using) that that section is a disk. The physical disk is real enough, but the part DOS sees isn't the actual disk, but rather something more like a virtual disk located somewhere on the media.

The task of creating a partition is really an exercise in creating a virtual disk in much the way you can set aside a block of RAM to make a RAM disk. It is, like a RAM disk, really more of an illusion, but it's the foundation on which our entire data structure rests—just this very tiny file.

What MIRROR's /partn switch does, then, is make a backup copy of this data file so if one of the worst imaginable hard disk problems happens—if the partition table becomes corrupted—UNFORMAT can write the backup copy over it. With any luck, you're right back in business that quickly and that easily with *no* lost data.

Of course, you can't store the backup copy of the hard disk partition *on* the hard disk—you could, of course, but if your hard disk went down you'd never be able to find your backup copy (or anything else). So MIRROR saves it to a floppy disk, prompting you with the following message:

```
Disk Partition Table saver
The partition information from your hard drive(s) has been read.
Next, the file PARTNSAV.FIL will be written to a floppy disk.
Please insert a formatted diskette and enter the name of the
diskette drive.

What drive? A
```

Unlike FAT and root directory backups that should be updated frequently and *must* be updated immediately following any unfragmenting or disk optimizing

operation, the partition table backup should be good until and unless you should ever repartition your disk for any reason.

For myself, I like to keep such data on a special bootable floppy. It doesn't have to be fancy—I just format with the /s switch and put a copy of COMMAND .COM on it, and it's bootable. You don't need an AUTOEXEC.BAT or CON FIG.SYS for this job. Of course, you're going to need a copy of UNFORMAT to put your backup copy on the hard disk, so I keep that on the floppy too. And here it is, in Fig. 15-9. You want to keep that floppy in a safe place then—a very safe place. Your whole hard disk could depend on it.

```
 Volume in drive A is HD_PARTN
 Directory: a:\*.*
command.com      41765   8-15-90    3:33
partnsav.fil      2048  10-29-90   19:08
unformat.com     17504   8-15-90    3:33
      62,464 bytes in 3 file(s)
     229,376 bytes free
```

15-9 Still another use of MIRROR is to save partition table data. Here's the directory of a bootable floppy containing a MIRROR partition table data file and UNFORMAT.

What does it all mean?

MIRROR, of course, is primarily for hard disks—the real MIRROR, that is. Its primary function is to save a copy of the FAT and root directory and, if you like, the partition table too. However, one of the nice things about the way DOS goes about saving critical system data is that you can use it on floppy disks as well as with your hard disk. This isn't to say that most users are likely to take the trouble to MIRROR all their floppies—or that I would even recommend it.

What I would strongly recommend is that you take advantage of the opportunity to try it out, to practice. Just take a floppy disk and copy files to it—either a fresh disk or at least one you know contains no files you really care about. Experiment with MIRROR on it—to get the feel of it. FORMAT and UNFORMAT it. In fact, run MIRROR on it (just MIRROR *d:*) and UNFORMAT without reformatting so, when disaster really strikes and suddenly it's not a drill, it's still old hat.

Above all, understand that what you're dealing with is a closely interrelated disk management package, rather just a bunch of new stand-alone utilities. Except for the new FORMAT.COM, not only are they not free-standing, but for the most part they're only fully functional when used collectively. Sure, the new DOS formatter is nominally nondestructive taken by itself, for all the good that does. But try to unformat a disk or even bring back any of the files it might have had before

reformatting with any other than the proper new DOS utility, and it just isn't going to happen as I showed you earlier.

So it's a totally self-contained package. Actually, about the only disk management feature Microsoft has left out this time around is a disk optimizer or unfragmenter—a little surprising, particularly because they mention disk optimizing several times in the documentation. They got most of it this time around, though. It looks like once they made up their minds to do it they really did it right.

Part 5

Beyond just DOS

MS-DOS is many things, and it's different things to different users. Nowhere is this more apparent than in the several major but more specialized utilities featured in this section, including the new DOSSHELL, QBASIC, EDIT, and DEBUG features.

And nowhere are the changes in DOS 5.0 more immediately apparent than in the all-new, user-friendly QBASIC interpreter or the new full screen text editor, EDIT. The shell has been considerably enhanced since it was first seen in DOS 4, as well.

DEBUG? Well it's still there. It hasn't really changed, but possibly your needs have and you therefore won't need it. But it's also included in this section, in case they have and you just hadn't noticed.

Part 5

Beyond Just DOS

16
CHAPTER

The new DOS shell

DOS first tried to add a shell in version 4.0, but like so many things in 4.0 it just didn't quite come off. 5.0 sports a new shell, similar in appearance to the old shell but with some brand new features, the most significant probably being the ability to switch back and forth between several different tasks.

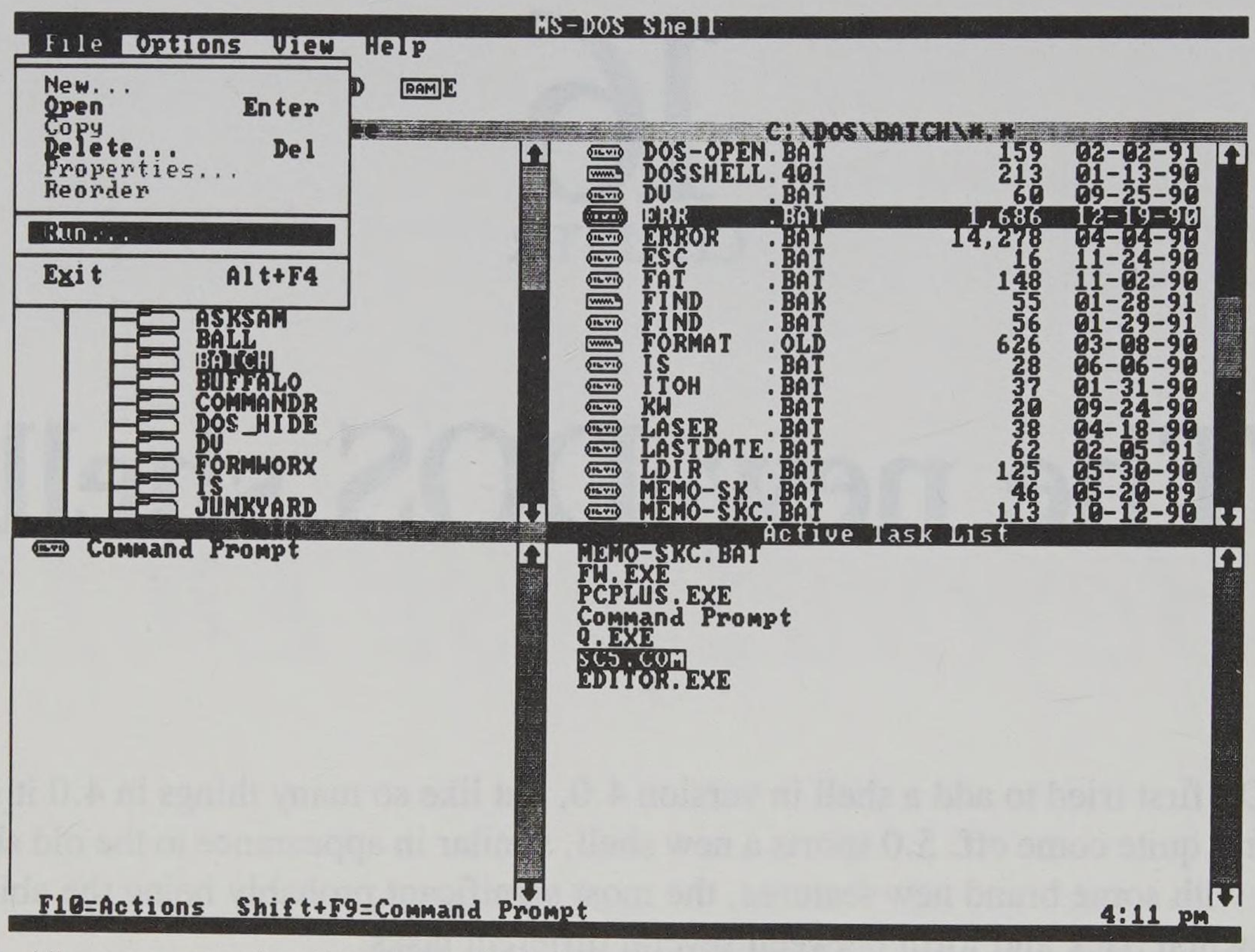
As you can see from Fig. 16-1, the shell is kind of like a one-stop shopping center. Everything you really need in DOS is there right at your fingertips—literally. You can use it all, or you can pick and choose the features that are right for your particular situation.

Running in either text or graphics modes—a choice you can make when loading DOSSHELL—you can have all five options shown on the screen in Fig. 16-1 in view and access any of the five, or narrow down the immediate access to either just the disk, directory, and file tree functions, or to just the main functions with or without the task switcher, depending on whether or not you've enabled it.

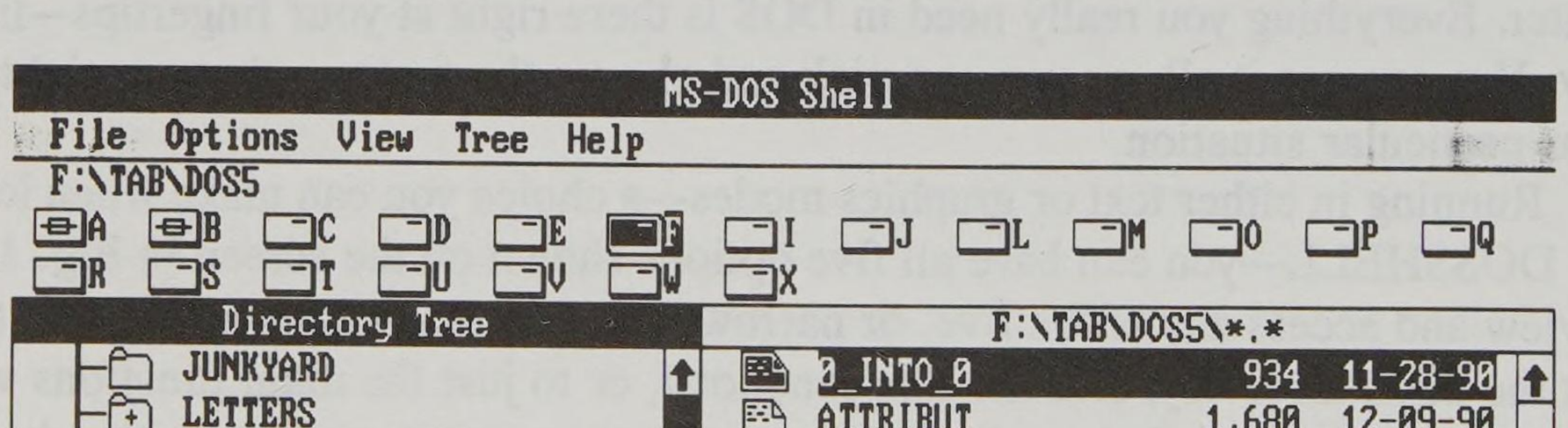
Or, in the file management mode, you can split the screen horizontally to display dual disk and directory listings simultaneously, bouncing back and forth between them at will—a handy feature when comparing the contents of two disks, weeding-out old files, and the like.

Or, if you don't need all the features but would, say, like to use it as a disk manager, you can devote the entire screen to that function, as shown in Fig. 16-2—even shifting easily to 43 or 50 line modes on supporting displays if you want. Or you could devote the whole screen to just the main functions menu or, optionally, the task switching menu.

At the risk of sounding like a snob, I'm really not all that impressed, although clearly the shell does have applications, particularly in the workplace where training new employees to use applications software can be headache enough without having to teach DOS as well. Otherwise, I suspect most users—almost anyone



16-1 Like a one-stop shopping center, the DOS shell makes pull-down menus, file management, task swapping, DOS commands, help, and other features available to entry-level users.



16-2 The screen can be devoted to file management, the task swapper, and the Main box, which is particularly useful for file management purposes.

with any DOS experience—will find, aside from being slow, that it's not only limited in what it does, but limiting in what you can do with it, and for my money they would have done better providing fewer features and doing more with them. But then I suspect the shell wasn't written with me or experienced DOS users in general in mind.

The new shell does add significant file management capabilities, something that's been badly needed in DOS and which has spawned a highly lucrative 3rd-party market topped by names like XTREE and PC Tools, both of which do everything the shell does and more with the exception of task switching. However, the shell that DOS provides is hardly on the same level as any of a number of 3rd-party shells that are available in my opinion—a disappointment viewed along side some of the excellent new 5.0 features like the new formatter, UNDELETE, UNFORMAT (as good, in my opinion, as any of the 3rd-party utilities on the market), and a host of other really significant enhancements.

Clearly, Microsoft didn't want to detract in any way from Windows sales with what they're giving you, essentially free, in DOS 5.0. And in that regard they've certainly succeeded, although whether or not that will generate any significant number of Windows converts or instead only spur sales of 3rd-party shells, task switchers, or full-fledged multi-taskers is problematical. Having said that, however, let's take a look at some of the new shell's features.

Modes and resolutions

The defaults for the shell would seem to be 25 lines in text mode with everything from CGA through VGA displays. This is easily changed to graphics mode on any display that supports graphics and to 43 or 50 lines by simply specifying graphics (/g) and, with supporting displays, by specifically calling for one of a higher resolution. The shell will come up in a 50-line graphics mode on a VGA if called up this way:

```
DOSSHELL /g:h 50
```

Applications run from the shell don't necessarily follow suit, however. Unless you've specified something other than 25 lines for the default display with a MODE command or other software capable of changing the default display mode, applications run from the shell will run in standard 25-line mode. ANSI.SYS must be installed from the CONFIG.SYS in order to change the number of lines displayed on screen with a MODE command.

The shell does impose a fairly stiff overhead on the system—40880 bytes. That's a fair chunk, but with HIMEM.SYS and about 47K of the DOS 5.0 kernel relocated in the HMA, the net result is still a net gain of roughly 6K of usable conventional memory workspace.

I found it singularly interesting that, although QBASIC and the 5.0 DOS SHELL support the use of a mouse, no mouse driver is supplied with DOS. This

is especially interesting because 5.0 seems a bit fussy about just what (or whose) mouse driver you're using. Not being a mouse enthusiast and, in fact, avoiding the pesky things whenever possible (which is most of the time), I did try using a mouse a couple of times out of curiosity and found that DOS 5.0 absolutely turned up its nose at the mouse driver I had. A warning box threatening all manner of possible dire consequences if I insisted on trying to use it. So much for that!

Fortunately, you don't have to use or even own a mouse to use the shell, and there are a number of shortcuts that generally enable you to bypass the menus once you've got the hang of it, making it possible for users with different skill levels to share the same machine and same working environment without penalty.

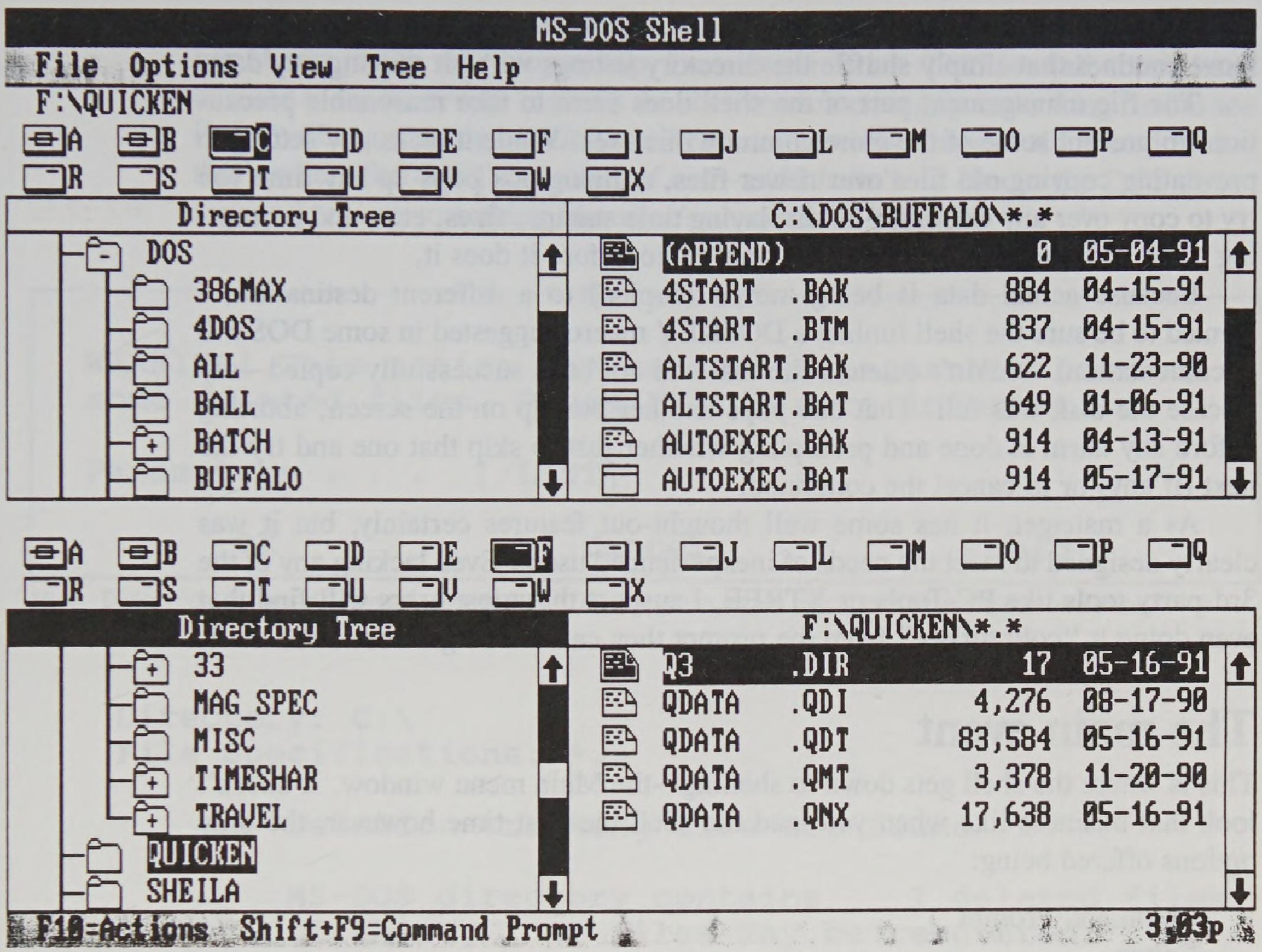
File management

As a file manager, the DOS shell works a little differently from XTREE and some of the other tree-oriented management utilities in several respects. It's also sloowww! To log or update the directory display for my hard disk, it clocked out at an average of something over 19 seconds compared to under 3 with XTREE on the same machine, in both cases starting with the program already loaded (about four times as long for DOSSHELL). That puts running the shell on a 16Mhz 386 about on par with XTREE on a 4.77Mhz 8088.

Like most, it allows you to SELECT files to be processed as a group. This is done more easily by clicking on the filenames with a mouse than from the keyboard, particularly if the files you're selecting aren't consecutively listed—and in all events a little clumsy alongside XTREE or some others. Instead of having a single key that you can use to tag files anywhere within the currently displayed directory to select consecutive files, you highlight the first one, press the Shift key and, while holding it down, use the up or down arrow keys to select as many or as few adjacent files as you want to be included.

If the files aren't adjacent in the listing, you have to press Shift−F8 to mark the first one you select, then arrow up or down using the spacebar to mark any others you want added. Change to a different directory to check something, and, unless you've selected the Select Across Directories option from one of the pull-down menus, any files you might have marked get deselected. Even if you've set the Select Across Directories option, changing to a different drive immediately deselects any files you've marked, as well.

There *is* a way around this, however. In the Dual File List mode, as shown in Fig. 16-3, you can switch back and forth between two different drives without disturbing your selections. This works only for viewing two directories. Even if files are selected in both windows, only those in the current (top or bottom) default are affected by group processing commands—COPY, MOVE, DELETE, etc. Also, even if you're already in the Dual File List mode, and, after selecting files for action in one window, you decide to log (or relog) onto the same drive in the other



16-3 The shell allows you to work with two disks or directories at the same time, but only one can be currently active.

window even just to get a different view, *that* deselects everything in both windows.

Head 'em up and move 'em out

The shell does have a MOVE facility I kind of like. DOS, of course, has never had a MOVE command and it still doesn't—not in the sense some other software "moves" files back and forth between directories by simply moving their directory entries, not the actual data sectors on your disk. The only way DOS "moves" files is by:

- Copying the files you want to move onto the destination disk and directory.
- Deleting the originals.

And guess what—that's just the way the DOSSHELL goes about it. This is more than just a subtle sort of "so what" nuance. The shell's method allows files to be

moved—singly or in groups—from one disk to another. This is not possible with move routines that simply shuffle the directory listings without moving any data.

The file management part of the shell does seem to take reasonable precautions to prevent some of the more common mistakes. While it lacks any setting to preventing copying old files over newer files, a dialog box pops up any time you try to copy over any existing file, displaying time stamps, sizes, etc., and prompting to be sure that's really what you want to do before it does it.

Because actual data is being moved (copied) to a different destination, I wanted to be sure the shell (unlike a DOSKEY macro suggested in some DOS 5.0 documentation) wouldn't delete a file that had not been successfully copied—say in case the disk was full. That one pops another box up on the screen, aborting before any harm is done and prompting whether just to skip that one and try the next (if any) or to cancel the command.

As a manager, it has some well thought-out features certainly, but it was clearly designed to meet the needs of inexperienced users. Even lacking any of the 3rd-party tools like PC Tools or XTREE, I suspect that most users will find that even doing it "cold turkey" from the prompt they can run rings around the shell.

The main event

This is where the shell gets down to shelling—the Main menu window. It doesn't look that business like when you load the shell the first time however, the only options offered being:

Command Prompt
Editor
MS-DOS QBASIC
[Disk Utilities]

Pursuing that to see just what "Disk Utilities" the shell will access, you'll find:

Disk Copy
Backup Fixed Disk
Restore Fixed Disk
Quick Format
Format
Undelete

All in all, a pretty limited selection unless you go to the command prompt, which defeats much of the purpose it would seem. Still when you stop to think about it, most DOS commands don't really lend themselves to running from the kind of shell environment the DOSSHELL creates anyway however—and, it turns out even the way the shell does some commands it does claim to support is open to some question.

Curious about how it would handle undeleting files, I tried that and the dialog box shown in Fig. 16-4 popped up. I wasn't sure what /LIST was all about, but I decided to try it anyway. It seems the shell (or something) goes looking first to see if you've been running MIRROR in TSR mode to keep track of all deleted files—that's where the /LIST comes in. If not—and I hadn't—it checks the DOS directory and reports something like what's shown in Fig. 16-5.

```
Undelete

WARNING! This action may cause the permanent loss of
some deleted files. Press F1 for more information.

Parameters . . .  [/LIST]

        OK                  Cancel                  Help
```

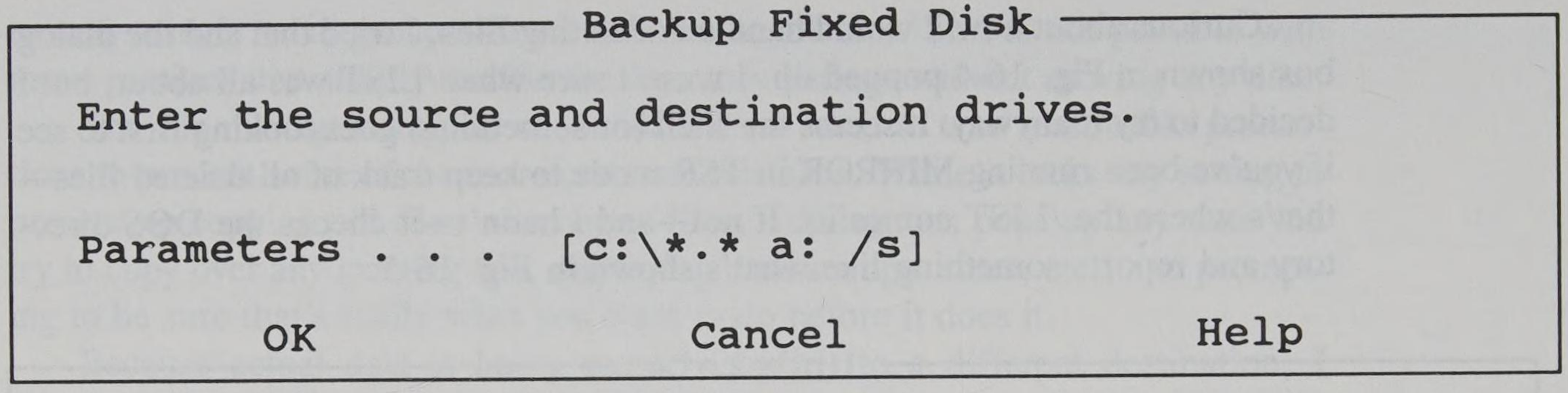
16-4 Dialog box displayed when you run UNDELETE from within a shell rather than from the command line.

```
Directory: C:\
File Specifications: *.*

        Deletion-tracking file not found
        MS-DOS directory contains    1 deleted files,
        Of these,    1 files may be recovered.

Using the MS-DOS directory,

        ?OXES                1051  2-18-91   9:13a   ...A

Press any key to return to MS-DOS Shell...
```

16-5 Running UNDELETE from within a shell is exactly like running it from the command line.

It doesn't seem to matter we just came from there and might like to go ahead and UNDELETE the file it says can be recovered. So to actually recover the file, I had to go to the command prompt (Shift−F9) and enter the UNDELETE command from there.

So I decided to see what the shell would do about backing up my hard disk, again approaching it from the standpoint of an inexperienced user. The box in Fig. 16-6 popped up, and while it would happily write over any files already on however many floppy disks you have to feed it to complete the job, DOS at least warns



16-6 The initial dialog box displayed when going through the shell to back up your hard disk.

of this fact before commencing (or continuing) with the following:

```
Insert backup diskette 02 in drive A:

WARNING! Files in the target drive
A:\ root directory will be erased
Press any key to continue . . .
```

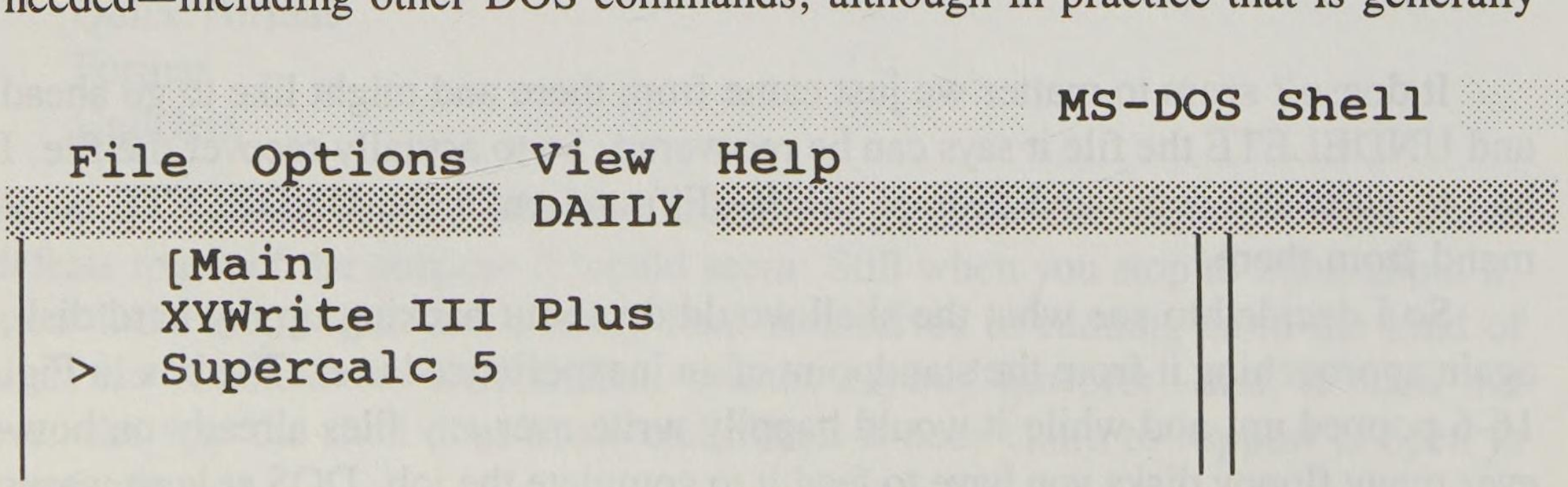
Otherwise, the default options, such as they are, are pretty straightforward and need little further explanation.

Beyond the basic defaults

The most important usage of the Main window menu would seem to be to set up the shell for safe and essentially goof-proof access to your applications software. This particularly in an office environment where average DOS skills would not likely be too high.

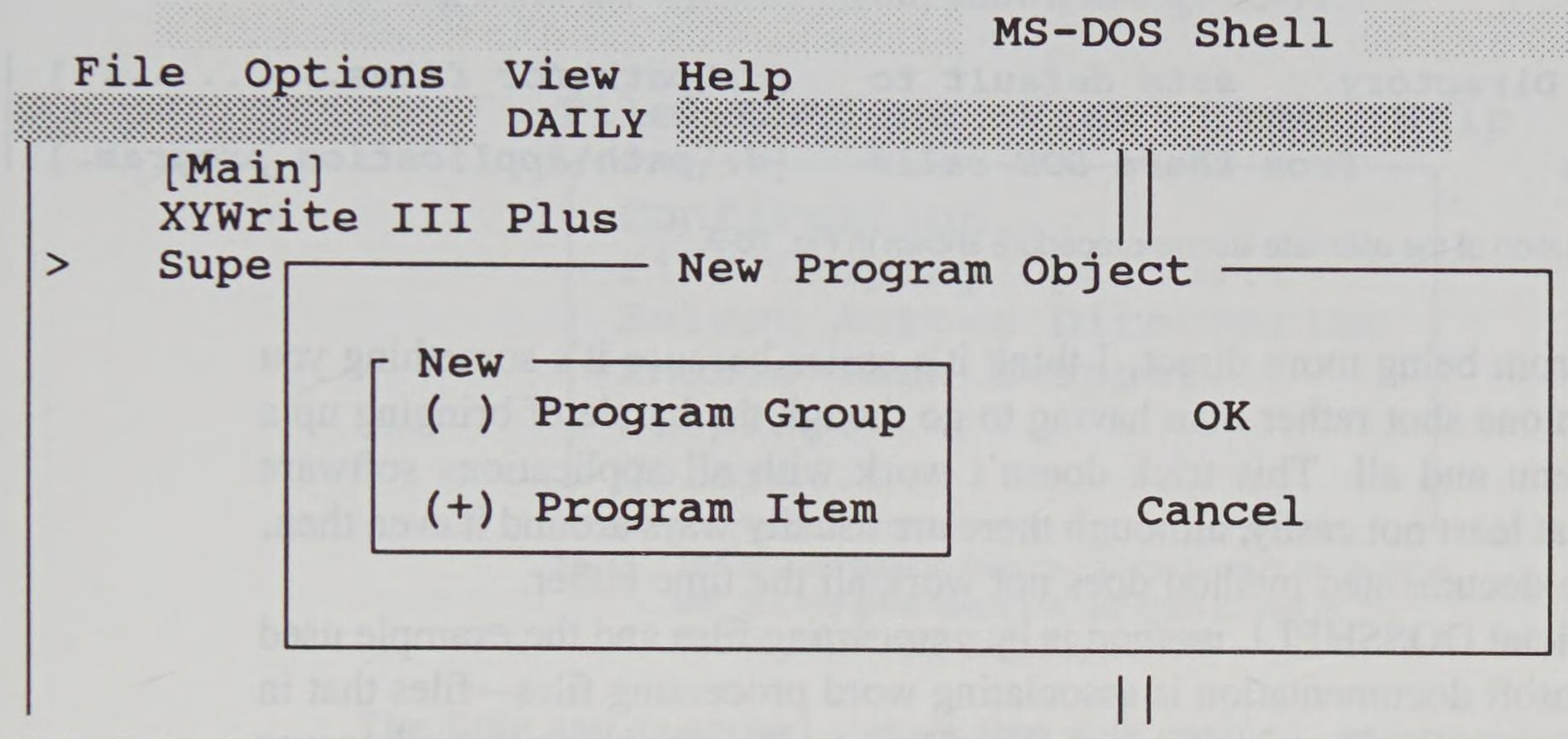
It is also here in the Main menu that you'll probably want to set up whatever applications you want to use with the task switcher on regular basis, as shown in Fig. 16-7, where I have added my word processor and my spreadsheet program by setting up a GROUP I call DAILY—which usually comes up as the default GROUP when I load the shell since the DOSSHELL seems to remember what group I was using last.

While only two are shown, you can keep adding new ones to the list as needed—including other DOS commands, although in practice that is generally



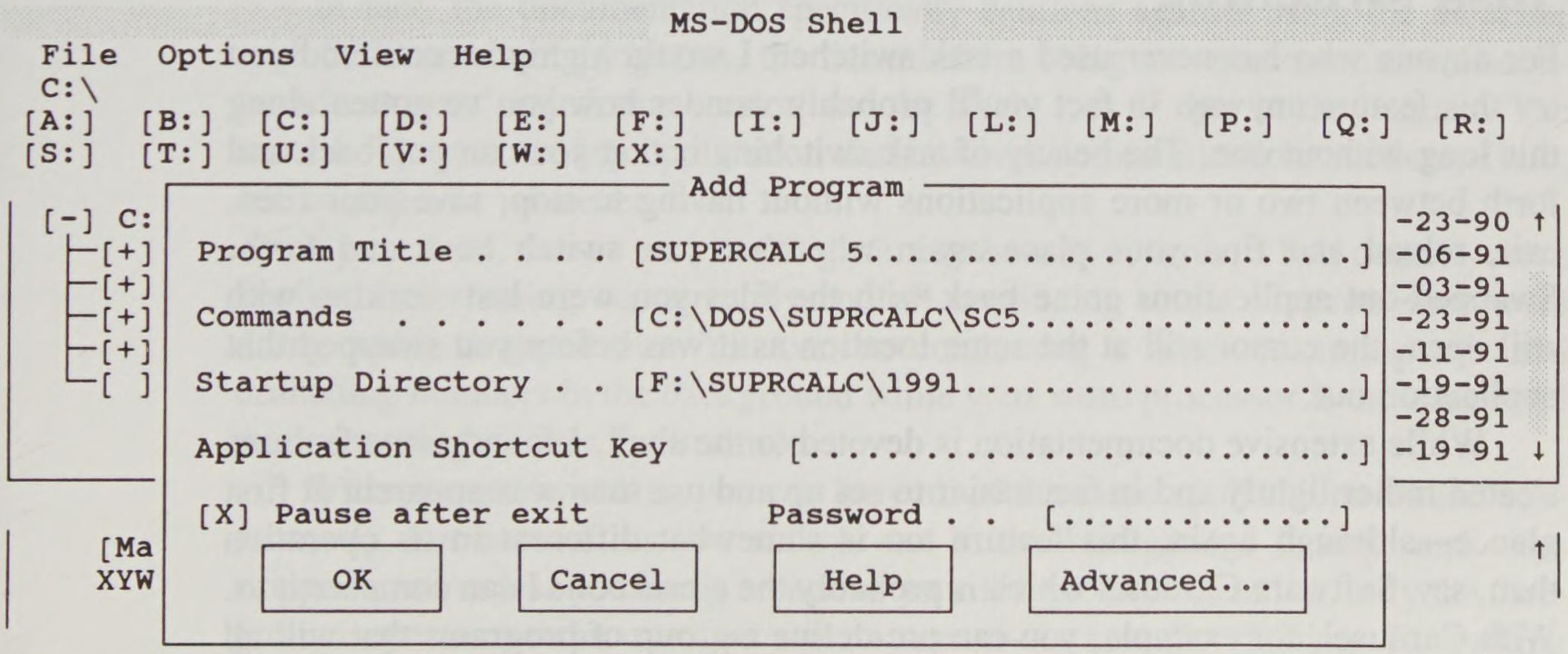
16-7 Applications displayed in the file group in the Main shell box.

not practical (see Fig. 16-8). What is lacking is a means of passing variable parameters. If the switches and/or parameters you would want to use are constant and can be pretty much chiseled in stone when you fill in the squares of the Add Program box, that's fine. Applications programs generally fit—or can be fitted—into that category, but most DOS commands do not.



16-8 The initial dialog box for adding additional programs to the group, for point-and-shoot access.

Again though, the DOS shell does not seem to be intended for DOS *users*—rather a non-DOS set. But clearly in a typical office environment the little time it can take in setting up the shell can pay off rather quickly. In Fig. 16-9, for example, I have set up the SUPERCALC 5 menu selection to not only call up the program but make the directory where spreadsheets files are kept the default directory.



16-9 A second-level dialog box for adding new applications to the group.

What I've done here differs from the documented way. It's a trick I use a lot with batch files though, essentially changing directories to the one that has the files, then calling up the application software from there as might be more apparent if you turn it upside down so you can see it from the point of view DOS sees it, as shown in Fig. 16-10.

```
Startup Directory    sets default to    [d:\path\for_files.....]
Commands             from there DOS calls   [d:\path\application_program.]
```

16-10 Explanation of the alternate startup procedure shown in Fig. 16-9.

Aside from being more direct, I think it's easier because it's something you can do all in one shot rather than having to go though the hassle of bringing up a different menu and all. This trick doesn't work with all applications software though—or at least not easily, although there are usually ways around it even then. But then the documented method does not work all the time either.

The official DOSSHELL method is by *associating* files and the example used in the Microsoft documentation is associating word processing files—files that in their example have .WP extension with WordPerfect, the idea being that whenever you open a .WP file the shell says "Ah ha!" and opens WordPerfect.

Borrowing from that example, all of SuperCalc's files have a .CAL extension—but so do the files of another totally unrelated application on my hard disk as well, and I'll guarantee you DOS can't sort out which is which. There is of course a way around this. There almost always is, but these are the times it's nice to have at least one other trick up your sleeve.

Task switching

For anyone who has never used a task switcher, I would highly recommend you try this feature anyway. In fact you'll probably wonder how you've gotten along this long without one. The beauty of task switching is that you can pop back and forth between two or more applications without having to stop, save your files, exit, reload and find your place again whenever you switch back and forth. Swapped-out applications come back with the files you were last working with still open, the cursor still at the same location as it was before you swapped that application out.

While extensive documentation is devoted to the shell, I found some features treated rather lightly and in fact easier to set up and use than was apparent at first glance—although again, this feature too is somewhat different in its operation than, say, Software Carousel which is probably the closest one I can compare it to. With Carousel, for example, you can pre-define a group of programs that will all load automatically at bootup. While opening programs with or without using the

task switcher can be reduced to a strictly point-and-shoot proposition, you still have to point and shoot to get this going.

Whether, having opened one program, you have to close your files and exit from the application or can simply put it in the background while you open up and use another at that point depends entirely on whether you have enabled task switching from the Options menu, shown in Fig. 16-11.

```
File  Options  View  Tree  Help
 Confirmation...
 File Display Option...
 Select Across Directories
*Enable Task Swapper
 Display...
 Colors...
```

16-11 The task-swapping feature can be turned on or off at will, but the shell saves the last state to use as the default when the system is rebooted.

The little star (asterisk) signals that it is enabled, and once set it will stay enabled until or unless you disable it—DOSSHELL remembers things like this and they become the defaults for the next time you run the shell.

This is *not* multitasking, however. It does *task switching* or, as it is often called, "context" switching. To switch from one task to another the old task is swapped, program code, open files and all, to temporary limbo on your hard disk. Unlike multitasking tasks that have been swapped out cannot continue running in the background, they are frozen and remain frozen until and unless they are loaded back into RAM to become the foreground operation once again.

In fact, the documentation specifically cautions against using the switcher with 3270 emulator programs, the hazard there being an unceremonious disconnect from your mainframe session and the likely loss of data in the bargain. The same holds true for any application that involves communication: modem, FAX, file transfer, or whatever.

By contrast, in true multitasking you can have two, three, a half dozen—more applications running concurrently and all continuing to do whatever they were doing when swapped out. You can be conducting a time-consuming sort and crunching numbers in the background while your word processor is on screen and you're writing a letter for instance.

With context switching you can have all of the same applications loaded concurrently, but as soon as you switch one to the background it stops and stays stopped until/unless you bring it back as your foreground operation: only the foreground operation—the one you see on screen—is actually running. Everything else is in a state of suspended animation.

This is perhaps easier to visualize in the way DOS manages task switching than with other context switchers such as Software Carousel because DOS does not use RAM when switching tasks. DOS writes the current contents of RAM—the last application you were using and whatever open files might be associated with it at that moment—to your hard disk.

Obviously, programs don't continue running when you save them to disk. Programs can only run in RAM—and even there only when the environment and other conditions are right. This also brings up the matter of speed because swapping to disk is a relatively slow operation typically taking 15 or more seconds even on a 16 MHz 386—longer even with larger programs as opposed to the near instantaneous switching times of better multitaskers and the significantly shorter switching times of context switchers such as Carousel that, unlike the new DOSSHELL can use expanded memory—if available—instead of having to go to disk.

So how slow is it? On a genuine old IBM PC with a vintage hard disk, typically about 25 seconds which, compared to the near-instantaneous full multitasking environment I'm used to working in. Fortunately, while no speed demon at best, given a faster hard disk on a faster machine the time required to swap one out and another in is reduced to a much more respectable figure—about four seconds on a 16 Mhz 386, for instance.

Put in perspective, these swap times are about on par with what you might expect with a context switching program like Software Carousel or even a multitasker like DESQview if not supported by sufficient RAM (including expanded memory) to handle everything in RAM—these two and other similar programs typically default to swapping to disk when they run out of RAM.

The difference, however, is that most other swap and multitasking programs use RAM first if it's available, swapping to disk only as a last resort. With any Multitasker—Windows, DESQview, or what have you—that makes swaps almost instantaneous. Even context switching in and out of RAM is pretty fast. Unfortunately the DOSSHELL swapper doesn't have that capability and likely won't in the foreseeable future.

Using the task switcher does impose considerable overhead on your hard disk—something that could be a problem for anyone like me whose always squeezed for hard disk space anyway. You might not even be aware of it until the day you suddenly run out of disk space in the middle of a session, particularly because, in its final release form, 5.0 tidies up after itself a lot better than it did in beta.

Still, if I might refer back to the way things were in beta, I think it will help to clarify this situation for although you won't see it this way on your disks the basic mechanism hasn't changed that much. Figure 16-12 shows what is, for me, a fairly typical work session with several applications open simultaneously. I suddenly found that DOSSHELL, in task switching mode, had created some 19 files taking up nearly 2 megabytes of hard disk space.

```
8F0EDOSC.BAT          25 .a..   1-29-91 10:33:26 am
9985DOSC.BAT          23 .a..   1-29-91 10:35:54 am
9F0DDOSC.BAT          24 .a..   1-29-91 10:37:12 am
A371DOSC.BAT          22 .a..   1-29-91 10:38:14 am
C109DOSC.BAT          25 .a..   1-29-91 10:45:10 am
C6A3DOSC.BAT          23 .a..   1-29-91 10:46:28 am
D0CBDOSC.BAT          24 .a..   1-29-91 10:48:50 am
DA62DOSC.BAT          27 .a..   1-29-91 10:51:06 am
 dos100d.tmp     470,314 r..h   1-29-91 11:00:18 am
 dos100e.tmp     374,170 r..h   1-29-91 11:00:26 am
 dos100c.tmp     224,138 r..h   1-29-91 11:00:48 am
 dos100b.tmp     193,594 r..h   1-29-91 11:02:52 am
5399DOSC.BAT           8 .a..   1-29-91 11:19:30 am
6423DOSC.BAT          26 .a..   1-29-91 11:23:22 am
DOSSHELL.INI      17,063 .a..   1-29-91 11:23:24 am
DOSSHELL.SWP      52,293 ra..   1-29-91 11:23:24 am
 dos1009.tmp     322,506 r..h   1-29-91 11:26:54 am
```

16-12 The task swapper is a messy housekeeper, leaving the hard disk littered with temporary files if applications aren't closed before exiting.

Provided you don't just pull the switch and close each application individually before you end the session, the shell cleans up the mess it's made, although you still have to have enough free disk space to accommodate the temporary files. Or alternately you can use TMP as a variable in your environment in order to use RAMDRIVE instead—provided you have a big enough RAM disk that takes care of the problem even if you get a little sloppy about closing out your applications.

Assuming that you don't use a RAM drive, with these swap files saved to disk, the obvious question in my mind was what would happen in case of a sudden shutdown—there is a crackling thunderstorm in the area as I sit here writing this.

Another near miss

Admittedly, you're not supposed to simply pull the plug to end a work session. Rather, you are expected to close out every task-switched application—it must be nice to live in such an orderly, unhurried world.

But say that last session was terminated by a power blip before you'd saved the files you were working on—even if the application you had running in the foreground and the latest updates to any open files were lost it would be nice to have the option of picking up the others where you left off. The idea is hardly new and there are programs that do support just such process restoration—the DOSSHELL does not, however.

Even so, any unsaved data that was in the temporary files if your last session had an unscheduled ending is, of course, still there buried somewhere on your hard disk—at least until DOS needs that disk space and writes over it. Still there

but not accessible—not by DOS at least, nor probably by most who use this feature. Any of a number of 3rd-party utilities such as the Norton Utilities can recover data from such files and save it to DOS files. When dealing with text files this is often satisfactory, but database and other applications files that use special formats may not be recoverable in this manner.

Still, despite the obvious limitations, I would suggest that anyone who has never used a multitasker or a context switcher at least try the DOSSHELL switcher. If nothing else it should open some eyes to the possibilities.

Beyond that, the new DOSSHELL would seem to raise as many questions as it answers. It does add a level of file management DOS traditionally has lacked—at least prior to DOS 4.0. As implemented in 5.0, it has been enhanced considerably (although the 4.x version did provide minimal internal mouse support). But even in text mode the shell is slow. I found it stiff and generally inflexible, and you won't even find it on my hard disk any more.

CHAPTER 17

Goodbye EDLIN, hello EDIT!

As many nasty things as I've said about EDLIN down through the years, it seemed only fitting that I use the new DOS EDIT feature to start this chapter by demonstrating how really useful this one is. It's not a full featured word processor by any means—it doesn't try to be. But it *is* a credible text editor with enough features to make it attractive for a lot of little jobs that don't require the horsepower of a full word processor.

While working on this book, in fact, I actually found myself using it quite often. It's ideally suited for those quickie jobs like writing or revising batch files, DOSKEY macros, the CONFIG.SYS file, etc., not only faster but easier than loading up a full-scale word processor.

Call it grandstanding to attract attention or whatever, that opening paragraph, in fact, was written with the new DOS 5.0 editor (EDIT) and merged with the text file for this chapter—a feat I never in my worst nightmare would have tried to with EDLIN.

I'm putting my cart before my horse here because, unlike poor old EDLIN, EDIT.COM is not a free-standing utility. It is really just a sneaky add-on to QBASIC and can run only under the QBASIC interpreter. No QBASIC, no EDIT. Because the mere mention of BASIC or anything approximating it is enough to put many users off, let me quickly add, however, that this relationship is totally invisible. You don't have to get involved with BASIC or QBASIC even.

Treat EDIT just the same as you would treat any ordinary DOS utility. On the command line, all you type is:

EDIT and Enter

You can enter the command as:

QBASIC EDIT [*filename*]

but it isn't necessary. Or you can do it this way:

EDIT *filename* and Enter

if you want to bring up a particular file (or open a new file under the name you specify). So I'm deliberately putting EDIT ahead of QBASIC in this book to disassociate it from the BASIC stigma.

Don't confuse it with word processing

As welcome an improvement as EDIT represents, it should not be confused with a word processor. It is not, despite the fact that many common word processor commands will work with it. What it lacks most is formatting and reformatting capabilities.

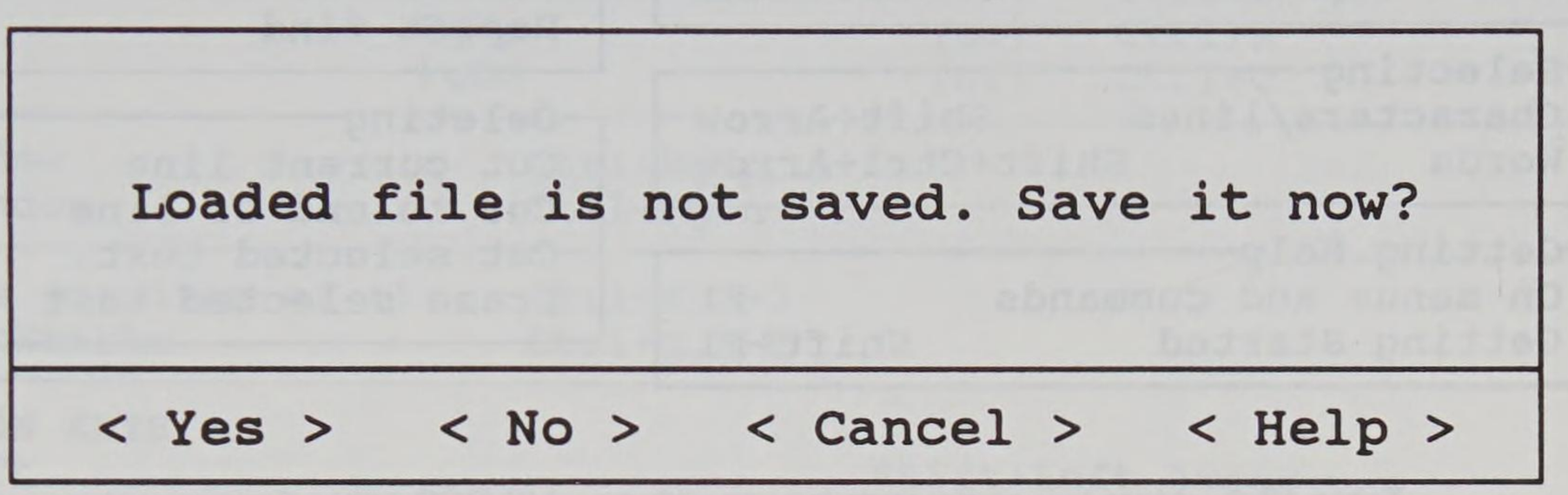
It does have one feature, however, that provides for setting and resetting left hand margins on the fly. This feature is not unique to EDIT, but is rather inherited directly from QBASIC, which does the same for programmers. It's sort of "monkey see, monkey do," but it works and is adequate for most jobs you'd use EDIT for, most likely. It works like this: tab or space in for the first line, EDIT tabs or spaces to the same point automatically when you start the next line and the next until *you* reset it. From then on—until you make another change—EDIT tabs or spaces automatically to match that setting until or unless you make another change. That's even slicker and a whole lot easier than many word processors.

Unfortunately, that's all the formatting help it gives you. EDIT has no provision for setting right-hand margins or, even more significantly, any wordwrap capability. When you want a line to break you have to break it with the Enter key, which puts a hard return into your file at that point. As a consequence, you don't want to try to edit any kind of document, because you'll have to chase down all the hard returns to the end of the paragraph you're editing—which might show on your word processor screen but not in EDIT. You must delete them individually, then reinsert them manually as required. This, of course, is not a problem normally when writing batch files and the like.

And EDIT has several big advantages over EDLIN for such tasks. First off, it isn't "line oriented." But more important, overall it is just naturally easy, really intuitive to use—anyone with even minimal computer experience should have little difficulty.

And EDIT offers help and then more help. EDIT even has some well thought-out safety features built in to protect you from your own mistakes—the most com-

mon ones, at least. Short of rebooting the system, you can't just dump EDIT once you've started working with it. Any time you have an open or unfinished file on screen—even one you've modified in any way since saving—the Exit command pauses, making you pick one of the four options shown in Fig. 17-1 before proceeding.



17-1 The new MS-DOS editor won't let you exit without saving an open file, pausing and providing you with a prompt.

If you chose the Yes (save it) option, the next screen prompts you for the filename and directory to save it to—in the case of a previously saved file, displaying that filename and directory as defaults.

A feature inexperienced users should find especially handy is a one-line explanation on the bottom of the screen for every option on the pull-down menus. These come up automatically and describe the currently highlighted choice—not in a lot of depth, but certainly a whole lot better than the sometimes cryptic menu entries. For example, when you pull down the EDIT menu, the first option is Cut Shift+Del. That's vague and tells you very little. But across the bottom of the screen you see *Deletes selected text and copies it to buffer*.

This clearly isn't a full fledged tutorial-type explanation, either—none of the help provided by DOS 5.0 fits that category. Still, it represents a giant step for DOS, and most users should have little or no need to look up the written documentation for this one.

A touch of the familiar

As mentioned earlier, the new MS-DOS editor recognizes keystroke combinations familiar to users of other Microsoft programs (such as Microsoft Word), as well as WordStar key combinations. However, for those unfamiliar with those command sets, EDIT provides another level of on-screen help, accessible at any time by pressing the F1 key, as shown in Table 17-1. WordStar keystrokes that can be used in the MS-DOS editor are listed in the right column of the three-column keystroke tables shown in Table 17-2.

Table 17-1 Shortcut keys for use with the new MS-DOS full-screen editor.

Shortcut keys

Inserting	
Switch to insert/overtype	Ins
Line above	Home,Ctrl+N
Line below	End+Enter
From Clipboard	Shift+Ins

Selecting	
Characters/lines	Shift+Arrow
Words	Shift+Ctrl+Arrow

Getting Help	
On menus and commands	F1
Getting Started	Shift+F1

Copying	
To Clipboard	Ctrl+Ins

Finding	
Search for text	Ctrl+Q,F
Repeat find	F3

Deleting	
Cut current line	Ctrl+Y
Cut to end of line	Ctrl+Q,Y
Cut selected text	Shift+Del
Erase selected text	Del

Table 17-2 More key commands for use with new MS-DOS editor. For convenience, alternate commands match those of several popular word processors.

HELP KEYS	
View Help on the MS-DOS Editor environment	F1 (or click the right mouse button)
Exit Help	Esc
View Getting Started	Shift+F1
Display the Help menu	Alt+H
Move cursor to next Help topic	Tab
Move cursor to previous Help topic	Shift+Tab
Move cursor to next Help topic with starting character	character
Move cursor to previous Help topic with starting character	Shift+character
View previous Help topic (repeat up to 20 times)	Alt+F1 (or double-click the Back button)
View the next topic in the Help file	Ctrl+F1
View the previous topic in the Help file	Shift+Ctrl+F1

CURSOR-MOVEMENT KEYS			
Character left	Left Arrow	(or)	Ctrl+S
Character right	Right Arrow	(or)	Ctrl+D
Word left	Ctrl+Left Arrow	(or)	Ctrl+A
Word right	Ctrl+Right Arrow	(or)	Ctrl+F
Line up	Up Arrow	(or)	Ctrl+E
Line down	Down Arrow	(or)	Ctrl+X

First indentation level of current line	Home	
Beginning of current line		Ctrl+Q,S
Beginning of next line	Ctrl+Enter	Ctrl+J
End of line	End	Ctrl+Q,D
Top of window		Ctrl+Q,E
Bottom of window		Ctrl+Q,X
Move to next window	F6	

Table 17-2 Continued

Increase size of current window	Alt+Plus		
Decrease size of current window	Alt+Minus		
TEXT-SCROLLING KEYS			
Line up	Ctrl+Up Arrow	(or)	Ctrl+W
Line down	Ctrl+Down Arrow	(or)	Ctrl+Z
Page up	PgUp	(or)	Ctrl+R
Page down	PgDn	(or)	Ctrl+C
Left one window	Ctrl+PgUp		
Right one window	Ctrl+PgDn		
Set bookmarks (maximum of 4)	Ctrl+K,0-3		
Go to set bookmarks	Ctrl+Q,0-3		
TEXT-SELECTION KEYS			
Character left	Shift+Left Arrow		
Character right	Shift+Right Arrow		
Word left	Shift+Ctrl+Left Arrow		
Word right	Shift+Ctrl+Right Arrow		
Current line	Shift+Down Arrow		
Line above	Shift+Up Arrow		
Screen up	Shift+PgUp		
Screen down	Shift+PgDn		
To beginning of file	Shift+Ctrl+Home		
To end of file	Shift+Ctrl+End		
INSERT AND COPY KEYS			
Switch between insert and overstrike modes	Ins	(or)	Ctrl+V
Copy selected text to the Clipboard	Ctrl+Ins		
Delete selected text and copy it to the Clipboard	Shift+Del		
Delete current line and copy it to the Clipboard	Ctrl+Y		
Delete to end of line and copy it to the Clipboard	Ctrl+Q,Y		
Paste the contents of the Clipboard	Shift+Ins		
Insert a blank line below the cursor position	End+Enter		
Insert a blank line above the cursor position	Home,Ctrl+N		
Insert special characters	Ctrl+P,Ctrl+key		
DELETE KEYS			
Delete one character to left of the cursor	Backspace		Ctrl+H
Delete one character at the cursor	Del		Ctrl+G
Delete the rest of the word the cursor is on			Ctrl+T
Delete selected text	Del		Ctrl+G
Delete leading spaces from selected lines	Shift+Tab		
FIND AND CHANGE KEYS			
Repeat find for same text	Ctrl+L	or	F3
Search for text	Ctrl+Q,F		
Change text	Ctrl+Q,A		

Again, what EDIT lacks most, it seems, is any real formatting capability. Like the electric bunny in the battery commercials, it just keeps going . . . and going . . . and going. Right up to DOS 5's 256-character line length maximum. No word wrap, no nothing.

Still, you can learn to live with things like that. In fact, EDIT would be a dandy alternative to always carrying around a word processor for a laptop if it wasn't at least as bulky as most word processors by the time you factor in the roughly 250K size of QBASIC.

Whatever its shortcomings, EDIT is a welcome addition in DOS 5.0. And as for EDLIN? Well, old dinosaurs do die hard it seems, and this one is still with us in DOS 5.0. If anyone really cares.

CHAPTER 18

QBASIC: more than just a pretty face

QBASIC is a lot more than just a prettied-up release of GWBASIC or BASICA. It's the next best thing to QuickBASIC—it even looks like QuickBASIC. In fact, it *is* QuickBASIC, except it isn't a compiler. Programs you write still wind up with a .BAS extension and will run only with the interpreter. It lacks a small handful of other features, too. But otherwise, it has the look and feel of QuickBASIC, and is a lot more likely to turn users on to BASIC programming than the old GW variety (which, although EDLIN lingers on, is gone from this release).

As a matter of interest, plain old BASIC was the first product marketed by Microsoft, that effort dating back to 1975. So there was a Microsoft before there was a DOS. Historically, the roots of BASIC go back even further, to the hall of academia, and, in its heyday, it played a major role in making the MIT Altair 8800 the first microcomputer to achieve commercial fame (if not success).

The world moved on, however, and though BASIC saw significant improvement in the more advanced BASICA and free-standing GWBASIC, it was slow, its convoluted branching often compared to spaghetti due to its heavy reliance on GOTO and GOSUB statements. Its ultimate fall from favor came with the memory explosion—the largest program possible was 64K.

QuickBASIC changed all that, offering programmers an easy upgrade to what some proponents call a "serious" programming language, without having to go back to square one with Pascal or C or some other unfamiliar language. And now QBASIC has made most of that greater power for the future—and yet compatibility with older BASIC/BASICA/GWBASIC programs—available to all DOS 5 users.

Whether or not you look at QBASIC as a serious programming language, it's certainly serious BASIC, weighing in a just over a quarter of a million bytes in MS-DOS, some three times the size of the GWBASIC furnished with DOS 4.0—this plus another 130,000 bytes of help. Of course, you can't judge QBASIC by the pound, but this is clearly not an upgrade. This is a whole new—new to DOS, at least—interpreter. And when Microsoft has you hooked, they'll gladly sell you the full compiler version. At the very least, this is about the neatest set of training wheels you're going to see.

The more things change . . .

It's been said, "the more things change, the more they stay the same." This certainly applies to DOS in many ways, and is, in fact, one of its greatest strengths. It's also no less true with QBASIC, although at first glance that might be more difficult to see because the user interface is visually so different.

If I were to try to sum it all up in one sentence, I think I'd have to say that QBASIC seems to preserve the best of the familiar, while quietly dispensing with many of BASIC's quirks—while at the same time adding power and a whole new level of responsiveness.

On the subject of responsiveness, one of the nicest things about QBASIC as opposed to plain old BASIC or GWBASIC is its immediate response to entry errors. If QBASIC wants a space inserted—say you type in:

```
locate 10,30 and Enter
```

The line will immediately change to:

```
LOCATE 10, 30
```

And yes, recognized command functions typed in lowercase are automatically raised to uppercase when you hit the Enter key, as well. Even nicer is the way it responds to many other common entry errors, like the one in Fig. 18-1.

And there's no waiting until you finish and try to run your program and have to go back through, line by line, to find your error. As soon as it detects an error in syntax, you get an immediate response—in the figure I had indeed omitted the closing parenthesis. One like this is pretty obvious and all the help you likely need is just a reminder.

Note that there are no line numbers. You can still use them if you want, but you don't have to. There are a lot of other little changes too, which, overall, I think make programing in BASIC not only less intimidating but actually attractive.

```
CLS : LOCATE 10, 1, 0
PRINT "ASCII Code : Character:  Scan Code  :";
DO
        DO
                response$ = INKEY$
        LOOP WHILE response$ = ""
        asciicode%=asc(response$
```

```
Expected: )

< OK >   < Help >
```

18-1 User-friendly QBASIC prompts immediately when entry error is detected—here I had omitted the closing parenthesis—taking much of the frustration out of programming.

More help

If you need more than just a little nudge, QBASIC has that too. In the case of Fig. 18-2, where the error seemed less obvious at first glance, I checked to see what HELP it had to offer and instantly got the response as shown in the figure.

```
HELP: Illegal function call

You are attempting to give an improper or out-of-range
argument to a Basic statement. Examples of such errors are:
  • A negative number is raised to a power that is not
    an integer.
  • A negative record number is given when using GET or PUT.
  • An I/O function or statement (LOC or LOF, for example)
    is performed on a device that does not support it.
  • Strings are concatenated to create a string greater
    than 32,767 characters in length.
  • A BLOAD or BSAVE operation is directed to a non-disk
    device.

ERR code: 5

< OK >
```

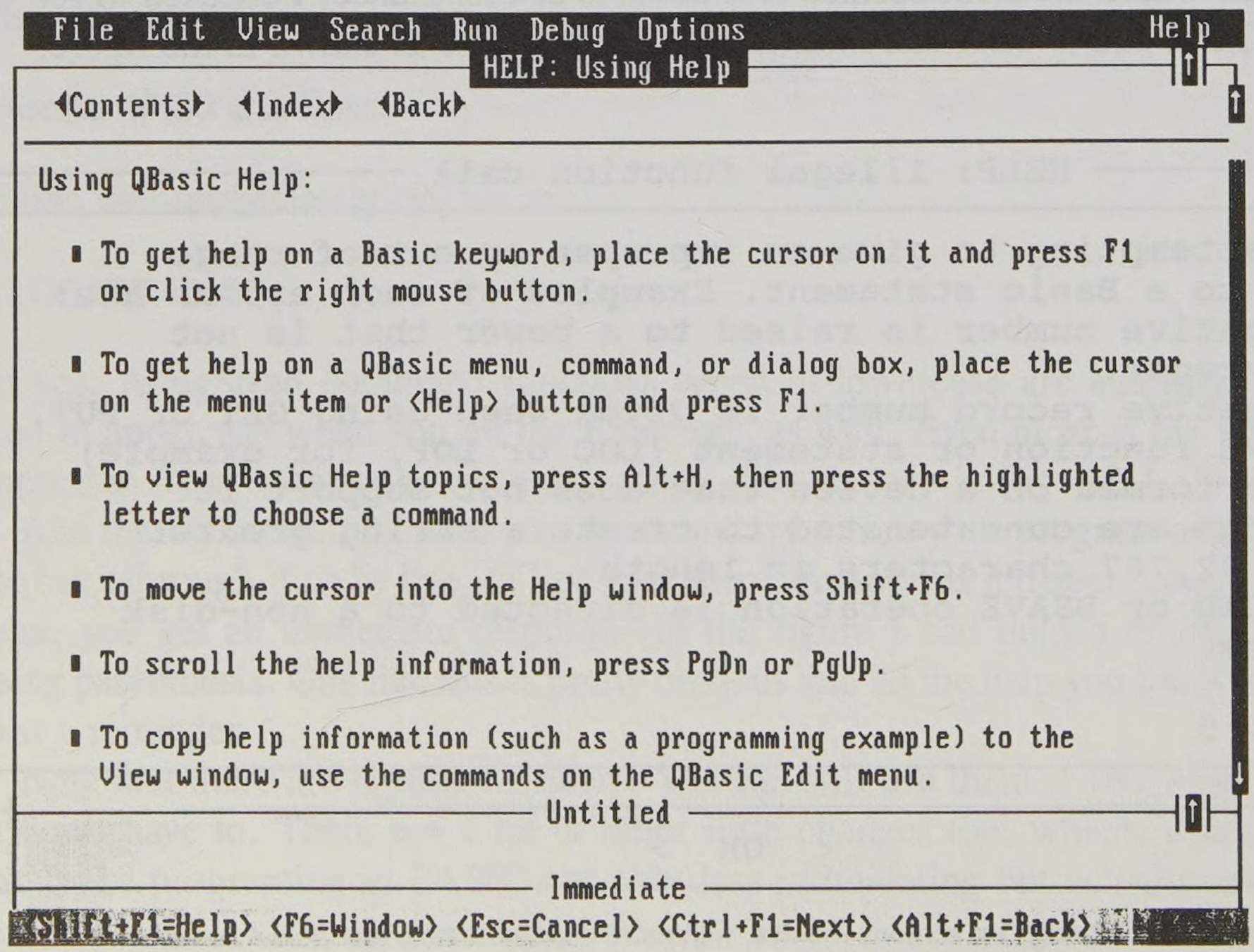
18-2 Choosing the Help option brings up screens like this, with more expansive help available if needed.

QBASIC certainly tries to keep you honest, and it succeeds in taking a lot of the frustration out of programming—especially for less than expert programmers like myself. As shown in Fig. 18-2, it has pointed to several possibilities it might have choked on.

Depending on your level of experience with BASIC, the help QBASIC gives you if you violate a rule may or may not suffice. You don't even have to wait to get into trouble to get help, however. From the moment QBASIC loads—before you even start to enter things—you have HELP at your fingertips.

The opening QBASIC help screen is shown in Fig. 18-3. Once into the help index (see Fig. 18-4), you access the contents under any letter of the alphabet by simply typing that key, then clicking on the keyword you need help for (or putting the cursor on it and pressing Enter). Figure 18-5, for example, shows a typical help screen—actually more than just a single screen of help, in this case for the IF statement.

Every keyword generates a similar screen. And QBASIC will display dialog boxes when you choose commands that have options. It even saves the last 20 help topics you viewed. You can review them by simply pressing Alt−F1 or by clicking the Backspace button repeatedly. So what's available on screen at the press of a key, then, is really the next best thing to the full-scale documentation that is *not* supplied.



18-3 The initial QBASIC help screen, which displays options and explains how to use the Help function.

```
File  Edit  View  Search  Run  Debug  Options                    Help
                    HELP: QBasic Online Help Index
 ‹Contents›  ‹Index›  ‹Back›

To get help on a QBasic keyword in the list below:
    1. Press the key of the first letter of the keyword.
    2. Use the direction keys to move the cursor to the keyword.
    3. Press F1 to display the help text in the Help window.

                                                                  A

  ABS Function                          APPEND Keyword
  ABSOLUTE Keyword                      AS Keyword
  ACCESS Keyword                        ASC Function
  AND Operator                          ATN Function
  ANY Keyword

                                                                  B

  BASE Keyword                          BLOAD Statement
  Basic Character Set                   Boolean Operators
  BEEP Statement                        BSAVE Statement
                          Untitled
                          Immediate
 ‹Shift+F1=Help› ‹F6=Window› ‹Esc=Cancel› ‹Ctrl+F1=Next› ‹Alt+F1=Back›
```

18-4 The QBASIC help index, which allows you to access information on any keyword.

```
File  Edit  View  Search  Run  Debug  Options                    Help
                    HELP: IF...THEN...ELSE Statement
 ‹Contents›  ‹Index›  ‹Back›

Executes a statement or statement block depending on specified conditions.

IF condition1 THEN
    [statementblock-1]
[ELSEIF condition2 THEN
    [statementblock-2]]...
[ELSE
    [statementblock-n]]
END IF

IF condition THEN statements [ELSE statements]

    ▪ condition1          Any expression that can be evaluated as
      condition2          true (nonzero) or false (zero).
    ▪ statementblock-1    One or more statements on one or more lines.
      statementblock-2
      statementblock-n
    ▪ statements          One or more statements, separated by colons.
                          Untitled
                          Immediate
 =Help› ‹F6=Window› ‹Esc=Cancel› ‹Ctrl+F1=Next› ‹Alt+F1=Back›
```

18-5 A typical QBASIC help screen.

Of course, if all else fails, Microsoft Press would just love to sell you one of their books on BASIC.

Expert systems

As much of an improvement as it is, someone who has never used BASIC isn't going to sit right down and call QBASIC up and start to work with it. It is a language and learning to use it is much the same as learning any language—French, Greek, Pascal, or what have you. This doesn't mean that you have to go to school to learn it. It can be self-taught, but you're going to have to buy a good book on BASIC basics—maybe two or three—to get you started.

Given that, and because the subject is so vast, about the best I can do with QBASIC in this limited space, and with an eye to the rather special group it's of interest to, is give something akin to a "jump start" to those with some experience in BASIC and, in particular, those familiar with QuickBASIC. Actually, if you're fairly familiar with GWBASIC or BASICA, you won't really need a whole lot of a jump start, but there are a few things you should be aware of.

Table 18-1 will give you some indication of the scope of some of the differences between GWBASIC and QBASIC. As you can see, there are many differences between the features offered by QBASIC and the BASICA of yore. While conceptually QBASIC was designed to be upwardly compatible with virtually all GWBASIC programs, there are enough differences to necessitate minor changes in some programs. So I think I should devote a little time and effort to that subject here—not just to the differences between QBASIC and the several BASIC variants that have traditionally been supplied with DOS, but with QuickBASIC too.

Converting other BASIC programs

When converting BASICA programs to QBASIC, there are several differences you should note:

- Make sure to save the BASICA program in ASCII format. QBASIC can't use programs that are saved in BASICA's binary format.
- Use the /MBF command option when starting QBASIC. The /MBF option enables QBasic to read and write BASICA data files.
- Change CALL statements to CALL ABSOLUTE. The BASICA CALL statement is equivalent to QBASIC's CALL ABSOLUTE.

Also, the following GW-BASIC/BASICA keywords are not supported in QBASIC:

AUTO	EDIT	MERGE	RENUM
CONT	LIST	MOTOR	SAVE
DEF USR	LLIST	NEW	USR
DELETE	LOAD		

Table 18-1 Differences between GWBASIC and QBASIC.

	BASICA/GW-BASIC	QBASIC
Language Features		
Line numbers	Required	Optional
Block IF/THEN/ELSE/END IF	No	Yes
Procedural blocks (SUB and FUNCTION)	No	Yes
User-defined types (records)	No	Yes
SELECT CASE	No	Yes
DO WHILE/UNTIL, LOOP WHILE/UNTIL	No	Yes
LONG (32-bit) integers	No	Yes
IEEE-format numbers	No	Yes
Constants	No	Yes
Fixed-length strings	No	Yes
VGA screen modes	No	Yes
Hercules/Olivetti support	No	Yes
Recursion	No	Yes
Cassette tape support	Yes	No
Code + data capacity	64K	160K
Debugging		
Single step	No	Yes
Procedure step	No	Yes
Trace animation	No	Yes
Breakpoints	No	Yes
Edit and continue	No	Yes
Editing		
Instant syntax checking	No	Yes
Diagnostic error messages	No	Yes
Instant help on errors	No	Yes
Multiple windows	No	Yes
Mouse support	No	Yes
Menu interface	No	Yes
WordStar compatible	No	Yes
Code organization	Linear	Modular
Document mode for text editing	No	Yes
Search/find/replace	No	Yes
On-line reference	None	Full

In addition, you might want to use REMLINE.BAS to remove unnecessary line numbers from the programs you're converting. QBASIC doesn't require line numbers, and removing them can improve a program's readability. The following QUICKBASIC keywords are not supported in QBASIC:

ALIAS	EVENT	LOCAL	SETMEM
BYVAL	$INCLUDE	SADD	SIGNAL
CDECL	Int86	Interrupt	UEVENT
COMMAND$	Int86X	InterruptX	

These differences seem to be fairly well documented in the on-line HELP, however. There are some other differences that seem worth noting, too. For instance, QBASIC seems to be limited to about a 160K program size, smaller than is possible with some of the more professionally oriented packages. This is still a significant increase in size from the old 64K limit, however, and frankly I can't imagine too many users that heavy into programming who wouldn't be using something stronger than QBASIC anyway.

Like several DOS 5.0-furnished features, QBASIC will support a mouse. You wouldn't expect a mouse to be included with the package, but you would think that they could at least include a driver (no mouse driver is supplied with Windows 3.0, either).

Overall, the difference between using the new QBASIC and the old stuff is so great you shouldn't even talk about them in the same breath. And yet it's still BASIC. But in a form that just could turn a lot of users on to BASIC.

CHAPTER 19

DEBUG: don't let it bug you

There's nothing really new in DEBUG in DOS 5, but I'm going to talk about it anyway. I used to wonder why so many writers spent so much time writing about DEBUG. It seemed so alien and so frightening—the very thought of changing things, of maybe screwing something up. But think about it—you change things almost every time you turn on your computer. You can create new files, change old files, and put new data in your spreadsheets or databases. Those things are changes too. Except those things seem safe somehow.

You know exactly what you've done if you go in and change a *two* to a *too*, or if you accidentally or on purpose take a letter out of the middle of a word. The world won't come crashing down around you if mess up this kind of thing, but DEBUG . . . sneeze and maybe change an *0A* to *0D*, and God only knows what *that* might do.

And it isn't a totally unfounded fear. But as a practical matter DEBUG is probably a lot less likely to actually change a value somewhere than you are by accident in any word processor, database, or spreadsheet entry. Those kind of programs are always waiting eagerly for input—they're written that way—and will save almost anything. DEBUG, on the other hand, is fussier.

You have to tell it specifically that you want to change something and precisely where you want to change it, and to what, before it lets you do it. And even then, just typing what might be changes to the screen is a good country mile short of actually changing anything. To make an actual change, you've got to do some other stuff, including giving DEBUG a WRITE (W) command.

So the W is probably the main thing you don't want to fool with until you learn your way around a bit—although I'll show you how to use it later in this chapter when I explain how to create a simple but useful little (13-byte) program with DEBUG's help feature.

There are some other places you can get yourself into minor difficulties with DEBUG—crash your system and stuff like that—and I'll try to point them out as we go along. But again, unless you've actually written some change to an existing file with the W command (and even that's only dangerous if you've done all the right stuff before you issue it), about the worst thing likely to happen is you'll have to reboot your system. Still, the risks are minimal.

I think perhaps that's what some writers tend to overlook in their zeal to show you all the neat stuff you can do. You can flap your arms and make like a bird, but that doesn't mean you're going to fly.

Another problem is that people tend to think of DEBUG as a tool only for programmers. It can be, but essentially DEBUG is an editor. Admittedly, it's a rather special kind of editor, but that's really all it is, and the sooner you come to see it in those terms the sooner you can settle down and start to use it comfortably.

Everybody's doing it

Well, not everyone perhaps, but there's an increasing tendency for the various computer magazines to offer little freebie programs, often cute utilities submitted by their readers. Often, all you have to do is type a few lines and *voilá!*—I'm talking mostly about short programs, not the kind requiring several pages of C or assembler listing aimed at someone who has lots more time than I have to diddle with them.

There was a time when BASIC was generally the medium of choice for users to recreate these little gems on their computers, but more and more it seems as though it's done with DEBUG scripts. Like the old BASIC/GWBASIC method, these go right to the heart of the matter—no cryptic mnemonics or "this does handstands" comments. Just the good stuff—hex stuff like A0, DF, and 1B. And you thought assembler listings were cryptic.

You shouldn't let it throw you, however, because there's really very little to go wrong—certainly not in any way that's going to hurt you. The worst that can happen is you'll find that the silly program doesn't work—which probably translates to "you made a typo somewhere when you typed it in." So you fix it and try again. No big deal—not once you understand what's going on. And scripts aren't that hard to understand.

Reduced to simple terms, *scripts* are really nothing more than a sort of batch file. Easily created—especially with the new DOS editor or any word processor that can produce straight ASCII files, a script is simply a list of DEBUG commands. The only difference is that instead of doing it step by step at DEBUG

prompts, you put the entire thing in a file, and use the file as the source for DEBUG. The nice thing about using a script is that, if you make a mistake and DEBUG won't swallow it, you don't have to go back and do the whole thing over. DEBUG will usually point out your errors one way or another and, if you're working from a script, all you have to do is go back and edit it and try again. And if you got it right that time, then you have got our program.

Actually, even the use of the term *program* is a bit misleading. Perhaps *file* would be the better word because it doesn't have to be a program to come within DEBUG's jurisdiction. It just happens that there are usually better, easier ways of dealing with other kinds of files. Usually, but not always . . .

Doing the unspeakable with DEBUG

Did you ever try to send a printer control string directly from your keyboard to your printer? They all start with an escape (ESC) character, which tells the printer not to print the string that follows but rather to interpret and act on its instructions. DOS, however, ascribes a rather different meaning to ESC, making it a pretty slippery character to deal with.

If you can get a handle on the slippery little critter, you can use ECHO and a little redirection to send something to the printer easily enough, and you could do it with a command like:

ECHO (*esc*)*string* >PRN

But as soon as you touch the ESC key, the command line clears, cancelling the command the way it's supposed to. If you try to get sneaky and hold down the Alt key while you type 27 (the decimal equivalent) on the numbers pad, DOS isn't fooled and shoots you right back to the prompt. You can slip one past DOS, however, if you do it from within a program rather than from the keyboard—the way printer drivers do—and the program doesn't even have to be all that fancy. Even a lowly batch file will do nicely.

The problem is that many word processors and editors don't do any better than DOS does when it comes to reproducing the Escape character (27 decimal or 1B in hex) as a character. So this is a good place to try your luck using DEBUG to create a little program—in this case, just a very simple one-line batch file, which, because most people have at least some grasp of batch files, should illustrate exactly what DEBUG does better than the abstractions of more serious programming.

My word processor, XyWrite, happens to be one that takes Escape characters right in stride, fortunately, so I'll start by using a DEBUG dump (d) to show you what I've actually got in the batch file I use and then duplicate it, as shown in Fig. 19-1. There's that pesky escape character—1B (I underlined it). The dump actually displays eight lines, but I've eliminated the rest for clarity here because this is

```
debug \dos\batch\esc.bat
-d
2009:0100  45 43 48 4F 20 1B 25 31-20 3E 50 52 4E 0D 0A 1A   ECHO .%1 >PRN...
```

19-1 Using DEBUG to examine a simple file for passing printer commands from the command line.

the entire batch file. You can forget about those last three lines—the ones displayed as periods at the right. So now you're down to just 13 characters. Even that includes a blank space or two you could eliminate, but let's go with that (see Fig. 19-2).

```
45 43 48 4F 20 1B 25 31 20 3E 50 52 4E
 E  C  H  O   esc  %  1     >  P  R  N
```

19-2 Side-by-side comparison of hex and ASCII characters. It's 1Bh, the Esc character, that's the problem.

If you now take the ASCII display at the right and match it, character by character, you'll see that at the level where DOS deals with things, the simple statement looks like so much gibberish. But if you compare the numbers with the numbers in the chart in Fig. 19-3, you'll see that, in every case, these are the hexadecimal equivalents of the ASCII characters you recognize as letters—the *hex* equivalents. These differ from the *decimal* equivalents (also shown) you might be more familiar with, but they in fact represent the exact same values—just expressed in a different numerical system.

D	H	Ch	D	H	Ch
64	40	@	80	50	P
65	41	A	81	51	Q
66	42	B	82	52	R
67	43	C	83	53	S
68	44	D	84	54	T
69	45	E	85	55	U
70	46	F	86	56	V
71	47	G	87	57	W
72	48	H	88	58	X
73	49	I	89	59	Y
74	4A	J	90	5A	Z
75	4B	K	91	5B	[
76	4C	L	92	5C	\
77	4D	M	93	5D	]
78	4E	N	94	5E	^
79	4F	O	95	5F	_

19-3 Partial list of decimal, hexadecimal, and ASCII equivalents. Only the hex values have meaning to your computer.

So let's take those 13 characters and write a script around them. For this, you can use any word processor or editor (including the new DOS EDIT) that can create a straight ASCII text file—yoou don't want any of the special control characters some word processors insist in slipping in.

First, tell DEBUG what name to give the file with an *n* command—you could do further down, but this way it's done and you can forget about it. Then with an *e* command, you can start to enter data.

```
n ESC.BAT
e,cs:0100 45 43 48 4F 20 1B 25 31 20 3E 50 52 4E
r,cx
d
w
q
```

The line of greatest interest is that *e* line. The comma is optional, inserted here just to separate it from the entries for clarity. CS is the default register that the *e* command uses, with 0100 being the starting address to use there (0000 is normally used only by .EXE programs). The colon is *not* optional. Otherwise, the line simply contains a listing of the hex values you want in the file you're creating.

The *r* command calls the CX register, and you have to tell it how many bytes you're going to write—D (hex) bytes in this case. The *w* command does the actual writing, and the *q* allows you to quit DEBUG. Be sure to include the *q* at the end of your script—and make sure you put a carriage return after it. Leave out either the *q* or the final carriage return and it will hang your system and you'll have to reboot.

Assuming you've got it right, then, to create ESC.BAT you call up DEBUG with this command:

```
DEBUG < ESC.SCR
```

Be *sure* you've got the arrow (<) pointing *toward* DEBUG. Point it the wrong way and DOS will obediently overwrite your script with garbage—no one ever accused DOS of being very smart.

If you've done it right (DEBUG <esc.scr), your screen should look like this when DEBUG runs your script and creates ESC.BAT—in this case, on your default disk and directory because you didn't specify otherwise:

```
-n ESC.BAT
-e,cs:0100 45 43 48 4F 20 1B 25 31 20 3E 50 52 4E
-
-r,cx
CX 0000
:D
```

```
-w
Writing 0000D bytes
-q
```

To use ESC.BAT, assuming it's still on the default directory or in the path, just type ESC at the prompt, leave a space, and then fill in the string you want to send your printer, like this:

```
ESC X1
```

X1 on one of my dot matrix printers sets letter-quality mode. You can use strings to combine several different printer codes if you like, provided you don't type any blank spaces between them.

Scripts and more scripts

Not all scripts deal directly with the individual hexadecimal characters. More than simply being an editor, DOS has the power to do limited compiling. Therefore, you might well encounter scripts that in fact *do* use assembly language mnemonics entered using the (assemble) command rather than the enter (e). These have a rather different look about them, as shown in the brief excerpt below:

```
MOV BX,AX
MOV AX,4403
INT 21
```

Often, you'll find mnemonic entries followed by plain English comments like this:

```
MOV DL,7        ; makes system beep
```

As long as the semicolon is there before the comment, DEBUG (or the Macro Assembler if you want to get a little fancier) doesn't even see them. When writing programs, it helps to put some of these little notes in as you go along—not when you write the program but when the program doesn't work just right at first. They're also pretty handy if you're trying to learn a little about programming. However, if you're copying a mnemonic script out of a magazine, you can leave the comments out—just type the part before the semicolon—to save yourself a bunch of typing.

Don't let it bug you

What I've tried to give you in this chapter is a glimpse of DEBUG, not to try to turn you into a programmer, but enough I hope to make you try it—if only just to make some cute little program you saw a script for somewhere. Like most things

in DOS, the biggest problem with DEBUG is simply overcoming the interia and ignoring that little nagging voice that says "I'd better not—I might screw up . . ."

I don't mean to say that DEBUG is for everyone who uses DOS. It isn't. But I'll venture to say that it's for a lot more than have ever used it. So don't let it bug you. Try it, and if you haven't you're almost sure to learn something—the more you use it the more you're going to learn.

Part 6

The whole nine yards

This is where I'm going to dump it all on you. This section contains the good, the bad, and the ugly—hidden files, system files, undocumented files, the works. Everything you're supposed to know about in MS-DOS version 5 is here, along with some things I think will be of interest.

Not a rehash of the Microsoft documentation, this section is intended as a adjunct to it. Uses and syntax are defined—in some cases more explicitly than in the regular documentation, but I've tried to put each command and feature into context so you won't think you're missing something if you're not using NLSFUNC or some other MS-DOS highly specialized, obscure, or in some cases antiquated but still supported features.

CHAPTER 20

Everything that's fit to print

With the exception of several new commands for use with networks only—commands that are neither covered in the regular DOS 5.0 documentation nor even installed on your machine unless networking is specified—in this section I've gone through each and every one of the commands in DOS 5.0. You'll note that this isn't just a rehash of the Microsoft documentation (much of which didn't even exist when this was written). It's intended primarily as a supplement to that documentation, exploring other usages and user questions that might not be addressed there.

To avoid confusion, I've generally observed the same conventions you'll find used in the Microsoft (or OEM) documentation—the uses of brackets, italics, etc. I'll review those conventions a little farther on, but before I dig in I'd like to point out some differences between the way you'll find DOS documented in this chapter and most other sources.

In the first place, this listing is entirely alphabetical, including all DOS commands or programs—even the hidden system files—and I've included all in a single listing rather than separating out device drivers and other special purpose items into separate listings you have to hunt for.

I've also included some information not usually included in such listings, such as the file extension for external commands—.COM or .EXE—or in the case of internal commands, whether they're internal to COMMAND.COM or IO.SYS. These bits of information aren't essential (except in specific cases where it's necessary to include the file extension on the command line), but I hope they'll assist you in understanding DOS and where it's really coming from.

The name of the command, as it's entered, and along with any extension, is located at the top of the page. The extension will be contained in brackets if it's required only in certain circumstances. Directly below is whether the command is external or internal, and any extension and specification. And below that, if applicable, is a comment, which I've used only to call attention to new or special commands. There are then eight short sections, as follows:

Function A brief description of purpose of command.

Usage Whether it's used from the command line or in batch files, exclusively in batch files, from CONFIG.SYS, etc.

Origin When the command became available in DOS or, if licensed, from whom.

Format COMMAND, followed by the correct entry syntax (including any for alternate usage).

Switches (also called parameters) A listing of all available switches and brief descriptions.

Errorlevel Supported or not supported.

Network use Supported or not supported.

Note Included only where special notation is needed.

Taking a closer look at the command syntax lines, there are so many possible variations on the basic theme that it's impossible to cover all the bases in a paragraph or two. But there is a general scheme to how commands are usually entered:

COMMAND_NAME [*d:*][*path*][*filename*]fixed[fixed][*variable*]

where brackets, [ ], always indicate an optional piece of information, such as:

- *d:* for drive if the target file isn't on default drive.
- *path* if target file is not on default or appended drive (or in path if file is executable).
- *filename* only required when command is to act on specific file (or group of files if wildcards are used).
- *variable* can be switches or parameters, usually chosen from accompanying listings.
- Items that must be entered exactly as shown are regular type, and are *always* required if not enclosed in brackets, and not always required if enclosed in brackets.

DOS is generally not case sensitive, allowing commands, variables, switches and other parameters to be entered in either upper- or lowercase, or any mix. I have—with the exception of a few rare instances where the documentation specifies uppercase for switches—used uppercase for command names and lowercase for everything else just to make the distinction more visible.

Some commands are shown with additional spaces added for the sake of clarity. All commands will work with the spacing shown, but in many cases spaces shown can be ignored. For instance:

filename | MORE

can be entered as:

filename|MORE

In general, however, I've tried to use spaces only where required and the syntax shown should serve as a suitable model in any case.

In the body of the text, I try to point out the weaknesses as well as strengths of each command. You'll also find some tricks included—like how to use some of the various commands quite painlessly in batch files and the new DOS 5.0 DOSKEY macros. You'll even find a trick or two that Microsoft says can't be done—like using wildcards with the FIND command.

DOS, as many of you already know, in many cases offers more than one command that can be used to do a particular job—COPY, XCOPY, and REPLACE to single out just three that come to mind. Where such overlaps occur, I'll try to call attention to them and point out specific differences that might help you decide which one is best for your needs.

I'll also look at the likelihood that most users will ever have occasion to use some of the more obscure commands. DOS offers a lot of things most of us will never use—things like support for foreign language keyboards and characters. Such things are of vital importance to a small select group of users, but most users might just as well know right up front that commands like NSLFUNC and CHCP are not for them. That not only helps to whittle DOS down to size and makes it less intimidating, but focuses attention on what is important to the larger user group.

If there's any message I would like to have come through this chapter loud and clear, it would be "make it easy on yourself." Some DOS users tend to think they have to fill in all the squares whenever entering commands, but nothing could be farther from the truth. Make it easy on yourself. Check the default values given and, unless your application specifically requires some value different from the default, let DOS do it.

Also, don't try to memorize all of the twists, turns, and quirks of syntax. The batch files and macros I've included here are only a sampling of what can be done. I've found that, in most cases, one trip to the documentation and some doodling with a batch file or macro can tame the most seemingly convoluted commands.

You will, of course, also find many things in the official documentation you won't find here. As stated before, this is intended as a supplement and not a substitute for that documentation. With a little time and effort on your part, and these two pieces of DOS documentation, you too can be an expert if you like.

ANSI.SYS

External (device driver)

Function Defines functions that change display graphics, control cursor movement, and reassign keys.

Usage Installable only from the CONFIG.SYS.

Origin Version 2.0.

Format DEVICE = [*d:*][*path*]ANSI.SYS

Switches

/x Enables the remapping of extended keys on 101 keyboards independently.

/L Retains the number of rows set by the MODE command after exiting applications. Some applications automatically reset the display to 25 lines on terminating, but this switch overrides such resets.

Errorlevel Not supported.

Network use Not supported.

Note No on-screen help available.

ANSI.SYS is one of those features some users swear by while others rarely or never use. I've gone through phases. It's not currently a part of my operating environment, in part because it's not compatible with some application software I use—it doesn't hurt, it doesn't crash the system if it's in there; it just doesn't work in the presence of some other software. So I don't bother.

You can do some very clever things with ANSI.SYS, as you might deduce for its origins—it dates to a time when DOS was young, and *clever* was whatever users could manage with the few tools available to them. ANSI.SYS was one of the best and certainly one of the most powerful of those tools. These days, clever programmers with far more powerful tools at their disposal write software packages you can buy right off the shelf that can do what ANSI.SYS can do—better in most cases. If not better, certainly more easily. I mean, what could be easier than having it handed to you on a disk.

ANSI.SYS is also one of DOS's most poorly documented features in several respects, in my opinion, which surely discourages many who might otherwise give it a try. It probably isn't of sufficient importance to warrant an in-depth study here, but a few basics would seem to be in order.

First off, while ANSI.SYS is a device driver, there is little you can do with it at the CONFIG.SYS level except just load the driver. There are, as noted above, a couple of switches that can be added to the command line in the CONFIG.SYS. But you can't pass parameters or commands to it at that level. That can only be done at the command line level or from a batch or other file that's accessible from

the command line. It's a two stage process: first you load the driver from the CONFIG.SYS and then access it from DOS.

There are several dozen basic commands, values, or parameters that can be passed to ANSI.SYS. Because these are detailed in the Microsoft (or OEM) documentation, I haven't repeated them here. There is a problem, however, with the way these commands—generally called *escape sequences*—are documented. They are strings preceded by the Escape character which (as in the case of sending special codes to printers) are used to signal the beginning of a string intended to be interpreted as special instructions for a device rather than printed or displayed as such.

In the manual you'll find ANSI commands documented as follows—for setting the screen (background) color and the text (foreground) color and attribute:

```
ESC[Ps;...;Ps]m
Set Graphics Rendition
```

The bracket ([) and lowercase letter (m) at the end are, in fact, requisite parts of the command, as might be expected, following the conventions used pretty much throughout the DOS manual. The ESC, however, is not. It represents the Escape *character*, which is an ASCII 27 (or hex 1B). It also can't be entered from the keyboard—at least not on the command line.

I must make a distinction here between the Escape *character*—and there really is such a thing—and what happens when you press the Escape *key* (or even if you try to outsmart DOS with an ASCII 27 on the keyboard by holding down the Alt key and typing 27 on the numeric keypad). You'll escape all right—right back to the DOS prompt, wiping out everything on the command line except the naked prompt itself.

Well, what about the letters ESC—like in the documentation? Forget it. Not even close. There are ways of doing it, of course, but unfortunately you won't find them in any of the official ANSI.SYS documentation I've seen so far from Microsoft.

Probably the easiest way to slip in something ANSI.SYS will recognize as an Escape character as such is tacked onto a PROMPT command. The PROMPT command, as you'll find in *its* documentation, supports its own proprietary set of codes including $e, which represents the Escape character. And even though the purpose of the exercise might not be to customize the prompt, it can provide a cheap-and-dirty vehicle for passing Escape sequences to ANSI.SYS.

To change your white-on-black default DOS screen to bright yellow on blue then (with the ANSI.SYS driver loaded), you can enter the following at the C> prompt:

```
PROMPT $e[1;33;44m
```

Reading from left to right after $e[, the number 1 calls for bold text on screen, 33

colors it yellow, and 44 paints the background screen blue—as indicated above, you can find these and many more values in the documentation.

Nothing much will happen immediately, but from this point on anything you enter will be displayed in bright yellow and the background will turn blue—but only on a line-by-line basis as you scroll down, until the whole screen background is blue. (The CLS clear screen command will change the entire screen immediately.)

Doing this without tacking on some sort of prompt string, however, will leave you with only a blinking cursor—not even a C> on your pretty new screen. You can do this either before or after the Escape sequence, or even sandwiched between two Escape sequences if you want to display the prompt in a different color, giving you these choices:

PROMPT *prompt_string* $e[.....
PROMPT $e[..... *prompt_string*
PROMPT *$e[..... prompt_string* $e[.....

Given the rather complex command structure this leads to plus the fact that you need to use the CLS command to clean things up suggest that this task can probably be best accomplished with a batch file, quite possibly the AUTOEXEC.BAT.

Working from a batch file rather than the command line might bring another player into the game, though, because many word processors and text editors have their own unique ways of dealing with the Escape character. A quick roundup of some but by no means all includes:

- XyWrite, the program I used to write this book, which displays a character on screen and inserts the 1B hex Escape character in the file when the Escape key is pressed.
- WordStar, in which you hold down the Ctrl key and press the P and then the Esc key, which might display ^[on screen.
- EDLIN, which provides a means for entering Escape characters in files with a Ctrl−V (^V) followed by a left bracket ([). This displays as ^V[.

Note, however, that the left-hand brackets shown in two of the above are *not* the left hand brackets required as part of the ANSI.SYS Escape sequences, but rather only part of the input code required to insert the Escape character. To enter the Escape sequence in the case of EDLIN, for example, you have to add a second left bracket, making the entry display something like ^V[[*string* on screen.

Some other word processors and editors allow you to enter the ASCII 27 character by holding the Alt key down while typing 27 on the numeric keypad. Generally, BASIC users can enter CHR$(27). Check your software documentation, however, as there are other ways that are unique to specific applications.

You can use batch files to pass commands to the ANSI.SYS driver by prefacing them either with the DOS PROMPT or ECHO commands—a CLS on the next

(or later) line will put color changes in effect immediately then. Another option is to simply put one or more ANSI commands in an ordinary ASCII file—just an ordinary text file, not even executable as a batch file would be—and then using the DOS TYPE command to "display" it on screen. Beginning with the line that starts the next DOS prompt, the new colors will take effect.

This is just a thumbnail sketch of a very complicated function, but it should at least get you past that slippery character that gets away whenever you hit the Escape key. You can't have any fun with ANSI.SYS until you get past that one.

APPEND

External (.EXE)

Function Similar to PATH except for data and other nonexecutable files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.2 and later.

Format APPEND *d:path*[;*d:*[*path*]][*switch(es)*]...
or APPEND (to display the current APPEND list)
or APPEND ; (to cancel the current APPEND list)
Note that a semicolon *must* be used as the delimiter between included directories in an APPEND string.

Switches

/? Displays on-screen help, including syntax, switches, etc.
/e Stores the APPEND path as environment variable.
/x Allows EXEC to use append search path to seek executable files that are version 4.0 and later.
/x:on Similar to /x above (default).
/x:off Turns off /x search and execute operations.
/PATH:on Files having drives and/or paths will be processed.
/PATH:off Files having drives and/or paths will *not* be processed.

Errorlevel Not supported.

Network use Supported.

APPEND is a good idea but still poorly implemented in 5.0. It's much like PATH and is similar in many ways, except that it's intended primarily for use in accessing non-executable files.

- You can include as many directories as you can fit on a 128-character line.
- You can access files in appended directories from *any* directory or valid drive without having to specify the path—not just from drives/directories included in the APPEND list.
- Writing a new APPEND list replaces the current APPEND, if any.
- A semicolon (;) *must* be used as a delimiter between directory entries.
- APPEND can be made a variable in the environment, although the benefit of doing so seems rather obscure.

APPEND would have been great back in the days of DOS 2.x. Today, however, you can usually do the job APPEND is intended for—and do it better—with decent hard disk management and simple batch files (and at a saving of 8360 bytes of RAM).

There are two major annoyances with APPEND, the first being that DIR doesn't display the files in appended directories. They're a lot like hidden files from that standpoint, although they don't fall in that category either because, even with its new ability to see hidden files, DIR can't find them that way either. To see what's there you have to call for a directory of appended directories the same as always; as if they were completely unrelated entities.

The biggest problem, though, comes when using APPEND with directories of text or data files—say, for your word processor or spreadsheet. Whatever drive or directory you append them to becomes the default drive or directory they'll be saved to, never mind that it's not where DOS found them. All of which tends to defeat the purpose of keeping things in separate directories in the first place.

Also, like PATH, APPEND is capable of dealing only with the *specific* directory or directories you've appended. It can't, for example, access files in any subdirectory of those directories. The disk in A: in this example is a file-by-file backup disk with the original directory structure re-created on the backup disk, XCOPY fashion. The first level on the floppy disk is FILES, with FICTION under it, so I tried:

```
APPEND A:\FILES;C:\
```

then with C:\ as the default, I tried:

```
TYPE \FICTION\MYFILE
```

and DOS returned:

```
Path not found - \FICTION\MYFILE
```

Knowing the file really was there, I changed my APPEND to:

```
APPEND A:\FILES\FICTION\
```

then again from the C>, I tried:

```
TYPE MYFILE
```

and that worked. Because APPEND, like PATH, will accept multiple entries, I could have included every directory on the disk in 128 characters, but changes to any of them would have been written to, in this case, the root directory of the wrong disk.

Rather than APPEND, I much prefer using batch files, like the model below, that accomplish pretty much the same thing with most of today's programs—and saves that 8K+ of overhead:

d: (drive where files are located)
CD *path*[*path*] (make file directory the default)
*d:**path**filename* (application)

Unless you've got multiple hard disks, you can probably forget the first line and cut right to the chase. Translated into real terms, it couldn't get much easier:

```
CD \FILES\EXCEL\1991
EXCEL\EXCEL
```

The first line makes the directory where I keep my current spreadsheet files the default, then I call the program up from there. I don't even have to clutter up my environment by including the program directory in the path. And any new files or changes to old ones are automatically saved back to the directory I want them in unless I specify something different.

The batch file approach doesn't allow the same simultaneous access to multiple directories—as many as you can fit on a 128-character line—that APPEND affords. But how often do you really need it for data files?

There are other factors that limit the use and effectiveness of APPEND. The DOS BACKUP and RESTORE commands can't be used when APPEND is in effect, and if in use APPEND must be removed prior to running either of the above.

The /x switch, when used, allows the DOS EXEC function to search the APPEND search path to find and load executables, as well. Unless you're given to using long PATH statement—which I strongly discourage—you could in fact probably use APPEND *instead of* PATH (but I don't recommend it).

The /e switch is interesting. This is the one that allows you to make your APPEND list a variable in your environment—although I have yet to figure out any real advantage to doing so. And it must be done in two stages for some peculiar reason known only to DOS. Step one is simply:

```
APPEND /e
```

You can't put anything else on the line that loads APPEND as an environment variable. Not even the APPEND list you want to put there—and, until you give it a SET command, it won't even show you that APPEND is loaded. You have to take a second cut at it to do that.

Note though that you don't have to use SET= when specifying an append path as an environment variable. Once you've set the stage with the /e command, the syntax is exactly the same as simply loading and using APPEND from the command line. The SET command entered without parameters will display the current APPEND list, as shown below:

```
COMSPEC=C:\COMMAND.COM
PATH=C:\DOS;C:\DOS\WINDOWS;C:\DOS\BATCH;C:\DOS
\UTILITY
TEMP=C:\dos\windows\TEMP
APPEND=A:\FILES;C:\
```

While APPEND can be used from within DESQview, it will typically return the message *APPEND/TopView Conflict*. This doesn't mean that APPEND isn't working or hasn't accepted the string you've given it, because sometimes it has (the APPEND /e can't be executed from DESQview, however). But because the DIR command doesn't read appended directories as such, it can be confirmed only by actually accessing something in an appended directory—typing a file you know to be there to the screen while in the directory you've appended it to, for instance.

Interestingly, the APPEND command can be used to append drives that have been assigned different drive designators with the ASSIGN command, and it can also be used to append directories on network drives. Overall, however, the drawbacks to date would seem to outweigh any advantage that might be gained from using APPEND. Particularly because its primary intended function can, at least in most instances, be satisfactorily simulated by other means.

ASSIGN

External (.COM)

Function Reassigns disk designation to another drive.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format ASSIGN [*w*=*x*] ([*x*=*w*]) (or) [*w*=*x y*=*z*]
or ASSIGN /?

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Supported.

Of greater use with early PCs before hard disks were in common use, this command was especially useful with some software written in such a way that it would load and run only from a specific drive—a practice that, mercifully, has virtually disappeared.

While ASSIGN does support network usage, there is some question about the future of this command. Microsoft, in fact, is currently recommending that users *not* use it, "in order to ensure compatibility with future versions of MS-DOS." This shouldn't pose a problem for any users currently using ASSIGN, however, because SUBST, a far more powerful utility, can, with minor exceptions, entirely replace it, as shown in the comparison below:

```
ASSIGN A=C
SUBST A: C:
```

Note that the syntax is slightly different for the two. In either case, any files on drive C: can be accessed as if on drive A:, or still accessed as being on drive C:. Such an assignment by itself is rarely useful, although in the example stated earlier—of software that could load and be run only from a specific drive—it was sometimes handy to temporarily completely reverse two drive designators with the command:

```
ASSIGN A=C C=A
```

Such a reversal isn't possible with SUBST, but for most users the need for any such skullduggery was pretty well past by the time DOS 3.0 came along.

In addition to simply supplanting ASSIGN in the context shown, SUBST adds considerable versatility by enabling users to substitute fake drive designators for complex routings, as shown here where T: is used to eliminate having to type

four directory levels to reach a particular set of files:

```
SUBST T: D:\TAB\DOS-MORE\BOOK\FINISHED
```

See SUBST later in this section for more details. Also see JOIN. ASSIGN should never be used in conjunction with BACKUP, RESTORE, LABEL, JOIN, SUBST, or PRINT. It should also not be used with FORMAT or DISKCOPY, as these commands will ignore it.

Unless you're still using some software that can't read and write file on disks in unassigned drives, you might just as well erase this one from your hard drive and forget it.

ATTRIB

External (.EXE)

Function Displays or modifies file attributes.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.0 and later.

Format ATTRIB [+/– *attribute*][*d:*][*path*][*filename*[*.ext*]

and format attributes are as follows:

+A	Sets the archive attribute.
–A	Clears the archive attribute.
+H	Sets read-only.
–H	Disables read-only.
+R	Sets read-only.
–R	Disables read-only.
+S	Sets read-only.
–S	Disables read-only.

Switches

/s Processes files in current path and subdirectories.

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Supported.

This command was greatly enhanced in 5.0 when the system and hidden file attributes were made accessible, as well as the more benign archive and read-only bits, but this doesn't mean that there is any less risk in tampering with these attributes than in the past. Rather, Microsoft has seemingly surrendered to pressures from various 3rd-party utilities that have made all attribute bits readily available for some time.

As a practical matter, however, other than read-only bits, there are few occasions when the average user needs to tamper with attributes. The Read-only attribute is an excellent way to protect frequently used text-format templates, "boiler plate" paragraphs, or any files used for reference only from inadvertent alteration. Here, in another typical application, I've made all of the files in my spreadsheet directory for a previous year read-only (using wildcards) to prevent them from being accidentally erased or altered while still keeping them available for ready reference:

```
ATTRIB +R \SPREADSHT\1990\*.*
```

Network users might find the Read-only attribute handy, too, because it can be

used to limit access to sharable files in networks situations that would otherwise allow both read and write access.

The A (Archive) attribute is normally set by DOS (if not already set) whenever a file is modified. This, however, is usually of little interest or importance except that the attribute is read by such other DOS utilities as BACKUP, XCOPY, and REPLACE, and might be used by them to determine which, if any files, to act on. In most cases, it's just as well to let DOS do whatever it wants with this one, although there are, of course, some exceptions.

Beyond that, most users would do well not to take advantage of the newfound power to change Hidden or System attributes. Prior to 5.0, the +H attribute could be set to hide files from inadvertent snooping (except with 3rd-party utilities), while still keeping them available for ready access by most programs. The enhanced 5.0 DIR command can see right through that with the /a:h switch, however.

One of the few occasions when it might be desirable to remove either the Hidden (—h) or System (—s) attributes is when you're removing old files left on your hard disk by certain programs you no longer use. You might not even know that they're still taking up many kilobytes of precious disk storage space.

Most people generally try to keep pretty good track of that kind of thing, but a quick check of one of my hard disks (DIR C:\ /a:h) while writing this book revealed the almost 400K of garbage shown in Fig. 20-1. None of these files showed in a normal (default) directory listing, but, aside from the first two (not in

```
Volume in drive C has no label
Volume Serial Number is 220C-4456

Directory of C:\

IO         SYS      33518 08-15-90    3:33a
MSDOS      SYS      37376 08-15-90    3:33a
AUTOEXEC   OS2        112 10-30-89    9:36a
CLOCK01    SYS       3086 09-21-89    5:37p
COMMAND    OS2      26268 02-20-89   12:00p
CONFIG     OS2        628 10-30-89    9:36a
DBMON      COM       4790 09-21-89    2:37a
DISK01     SYS      29111 09-21-89    5:37p
KBD01      SYS      15480 09-21-89    5:37p
OS2KRNL            283493 09-21-89    5:37p
OS2LDR             12710 09-21-89    5:37p
PRINT01    SYS       7925 09-21-89    5:37p
SCREEN01   SYS       1640 09-21-89    5:37p
      14 file(s)     456186 bytes
```

20-1 The new /a:h switch reveals nearly 400K of forgotten hidden files an ordinary listing wouldn't show. These were inadvertently left behind when deleting old OS/2 installation.

bold print), all were left behind by a previous OS/2 installation, even after I had cleared out and removed several whole directories of other OS/2 files. Changing the attributes on these (—r—s—h, because some had everything set) allowed me to delete not only these files, but some leftover odds and ends in several other directories, as well.

A word of caution, however. Before removing any suspect files completely, it's a good policy to copy them to some temporary holding area first—someplace where DOS can't find them, but where you still can if you find you really need them. I have a JUNKYARD directory. That's where these went, and the system ran just fine without them. I also know where to go the next time I need more hard disk space. It's like found money.

Except for an occasional housecleaning, you're better off to leave the System and Hidden file attributes alone—and be careful of the wildcards when changing any of them and erasing any files that might have had them set. Still, used with discretion ATTRIB can often be a handy thing to have.

BACKUP

External (.EXE)

Function Backs up one or more files to a different disk.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format BACKUP *d:*[*path*][*filename*[*.ext*]] *d:*[*path*][*switch(es)*]
or BACKUP *d:*[*path*][*wildcard(s)*] *d:*[*path*][*switch(es)*]

Switches (not all switches are available with all versions of DOS)

/? Displays on-screen help including syntax, switches, etc.
/a Adds new backup files without erasing old files.
/d:*mm/dd/yy* Backs up only files that were modified after a specified date.
/f:*size* Formats target disk (see FORMAT /F:).
/l:*filename* Creates log of files backed up in the filename specified.
/m Backs up modified files only.
/s Backs up subdirectories.
/t:*hh:mm:ss* Backs up only files modified after a specified time.

Errorlevel

0 Process completed, terminated normally.
1 No files found to back up.
2 Some files not backed up due to sharing conflicts.
3 Execution terminated by user (Esc or Ctrl−Break).
4 Execution terminated due to execution error.

Network use Supported.

While BACKUP has seen some improvements through the years, it shows little if any improvement over version 4.x. For users with a little patience, limited resources, and rather limited needs, BACKUP is generally adequate, however. And actually what most users see as weaknesses can be overcome with fairly simple batch file techniques.

Files created by BACKUP are in "archiving" format—not to be confused with archiving as used sometimes to describe certain data compression techniques. In the DOS context, archiving is essentially a process of appending all included files into a single BACKUP file. Unlike files archived with data compression techniques, any legible ASCII text or data remains legible and recoverable with the Norton or Paul Mace utilities should the file become corrupted. Still, legible or not, files saved by BACKUP can't be used without first using RESTORE, a separate DOS utility, to separate them into individual files again.

While BACKUP allows the use of all DOS wildcards, they're often too broad

and you might want to add lines to your BACKUP batch file for specific directories or groups of directories. For example, in the batch file below, because I want to limit my backup to just four directories on a disk that has four times that many at just the first level, I've included one line specific to each:

```
BACKUP D:\FICTION\*.* B: /S/M/L:B:LOG
BACKUP D:\SUPRCALC\1991\*.CAL B: /A/M/L:B\LOG
BACKUP D:\QUICKEN3\QDAT*.* B: /A/L:B:\LOG
BACKUP D:\PROJECTS\*. B: /A/S/M/L:B:\LOG
```

I've used several different tricks here to customize my backup. In line one, by using the wildcard *.*, and then the /s and /m switches, I've including every file that's new or modified since the last backup, including any in the subdirectories under FICTION.

On the next line, I've narrowed the search to new or modified files in one particular subdirectory only, and then to only those with a .CAL extension. In the next line, I pick up all files that meet the search criteria, whether they've been modified or not. And the last line picks up any new or modified text files that have no extension (eliminating any .BAK files), which works fine because I never use extensions unless I have to.

Note, however, that beginning with line two *all* lines use the /a switch so that their output will be appended onto the existing BACKUP file. This is an absolute *must!* Otherwise, BACKUP will assume that not only are any other files in general but even any previous BACKUP files no longer needed, and will wipe out everything—files, backups, logs, the works. This, if nothing else, makes it not only desirable and much easier but almost essential that you manage BACKUP from a batch file, which, once tested, is not subject to the kind of fatal typos we all make sometimes. Forget the /a in line four and you can kiss any other backup files on that disk—that day's or several weeks' worth—goodby!

The /l: switch is an absolute must if you want to keep your sanity. Without it you have no idea what's on any particular backup disk, particularly if you've used other switches limiting those backups to new or modified files only.

This still leaves you about a mile short of some of the slick 3rd-party backup utilities, but with a little razzle-dazzle, you can make your basic batch file act respectable and, with a little help from DOS's ERRORLEVEL, almost friendly. It's actually two batch files: the first one, a one-liner I call UPDATE.BAT, which calls the second batch file (DO.BAT), but more importantly passes a *drive*\ *directory*\ *etc.* list to the second file as parameters—up to a maximum of ten—as shown below:

```
DO d:\path\etc d:\path\etc d:\path\etc d:\path\etc d:\path
\etc d:\path\etc.
```

DO.BAT then passes these parameters (after checking to be sure it still has at least

one left to process) one at a time to BACKUP in line three of the following. To keep things from getting out of hand, DO presupposes that the backup disk will always be in the same drive, so it adds that information along with all the switches you want for a normal backup to the BACKUP command, along with the *drive* \ *directory* \ *etc.* it has to process. And with that, it's off and running.

```
:LOOP
IF !%1 = =!"" GOTO END
BACKUP d: %1/a/m/s/l:d:\LOG
IF ERRORLEVEL 0 ECHO Backup of files in %1 completed satisfactorily
IF ERRORLEVEL 1 ECHO No files found matching backup criteria in %1
IF NOT ERRORLEVEL 0 GOTO ERROR
SHIFT
GOTO LOOP
:ERROR
IF ERRORLEVEL 3 ECHO Backup aborted (Ctrl – C) before completion
IF ERRORLEVEL 3 GOTO EXIT
IF ERRORLEVEL 4 ECHO Terminated due to error backing up files in %1
IF ERRORLEVEL 2 ECHO Some files not backed up due to SHARING
conflict(s)
ECHO Use Ctrl-C to abort or press any key to continue
PAUSE|SHIFT|GOTO LOOP
:EXIT
```

Here I've added several more batch file tricks to come up with an automated backup that's tailor-made to handle backups that include as many as 10 possible inclusion options—SHIFT takes care of that by *shifting* your list of drive \ path \ etc. candidates to the left and replacing %1 until all options are processed. However, it checks the ERRORLEVEL returned by the last line each time before proceeding. From that, the batch file is able to branch to an appropriate action or offer options for each option.

If the ERRORLEVEL returned is 3 because you've aborted the program, you obviously want to exit to the DOS prompt at that point. With ERRORLEVEL's 2 or 4, however, you might want to continue. So after reporting which *drive* \ *directory* \ *etc.* failed to backup properly, you have a choice of terminating (Ctrl−C) or allowing the program to continue, in which case it *shifts* again to process the next *drive* \ *directory* \ *etc.*

Note again the exclamation point in the line that test to see if any search criteria are left to process (IF !%1 = =!""...). It's meaningless, but DOS is unable to compare nothing with nothing so you have to throw it a bone sometimes to keep it happy—come to think of it, it does look kind of like a bone, doesn't it? Even with all this modification, BACKUP isn't going to win any beauty contests. But it can do a job, and that's what counts.

Like several other DOS utilities, BACKUP should not be used when ASSIGN, JOIN, SUBST, or APPEND are in effect, as any of these can cause unpredictable results and a possible loss of data.

Users in the past have sometimes had a problem if they tried restoring backup files with the RESTORE utility from a different DOS version. Supposedly, Microsoft has fixed that now. DOS 5.0, they claim, will restore backups made with earlier versions without problems. It is something that upgrade users should, however, keep in mind.

BREAK

Internal to IO.SYS, COMMAND.COM

Function Sets or clears Ctrl−C (or Ctrl−Break) checking.

Usage Most often included in the CONFIG.SYS file when used at all, it can also be turned on or off from the command line or a batch file.

Origin Version 2.x and later.

Format BREAK [*on*][*off*] (from the command line)
or BREAK=ON (in the CONFIG.SYS file)

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Not supported.

Intended as a safety net, BREAK gives you the opportunity to interrupt or terminate a process (other than rebooting) by pressing either the Ctrl−Break or Ctrl−C key combinations. When successful, BREAK returns you to the DOS prompt in order to do whatever you need to do there—restart the process just terminated or some completely different task.

BREAK is initialized when you start your system, the default being off. Even in its default state, BREAK still checks for Ctrl−C during screen, keyboard, and printer reads and writes. The command line BREAK=ON can be included in your CONFIG.SYS if you want DOS to check for Ctrl−Cs during disk reads and writes as well. Or the BREAK state can be altered from the command line later, if desired.

Regardless how or when you do it, you should turn BREAK on only with caution. Likely problems with BREAK are not so much the fault of DOS, but other software developers who don't take the possibility of users breaking out of programs unexpectedly into account. Breaking out of an application is different than breaking out of a batch file—there, at least, the system usually pauses and questions whether you want to *Terminate batch job? (Y/N)*. The danger lies in the fact that the stop is so sudden and unceremonious. Breaking out of a running application can lead to system crashes with some software. Worse, BREAK=ON can actually dump some programs and leave files open, databases corrupted, etc.

Careful programmers generally restrict DOS's opportunities to check for BREAK commands. Even if your software can weather the hazards of unanticipated breaks, at the very least turning BREAK=ON significantly increases system overhead, making applications run slower.

This is not to say that BREAK doesn't have a useful role in DOS. It does. It can be of great help to programmers while developing and debugging programs. In any case, the best advice is to take the default on this one unless you're sure what you're doing.

BUFFERS

Internal to IO.SYS

Function Allocates RAM for buffering (temporary storage) of most recent disk read/write data.

Usage Installable only from the CONFIG.SYS file. Some 3rd-party software provides executable files that allow the number of buffers to be set either from the commands line or in batch files. The MS-DOS BUFFERS command, however, doesn't allow that kind of usage. /? displays on-screen help, including syntax, switches, etc.

Origin Version 2.x and later.

Format BUFFERS=*nn*[,*mm*]

Switches None. The /x switch supported by 4.x does not seem to be supported in version 5.0. That switch, which places the buffers in expanded memory, might have been deemed unnecessary because, in version 5.0, the buffers are automatically loaded to the high-memory area when the DOS kernel is loaded there using the DOS=HIGH command option from the CONFIG.SYS.

Errorlevel Not supported.

Network use Not supported.

Whenever DOS reads from or writes to the disk, it uses portions of RAM designated as *buffers* in order to store the data. Each buffer holds the contents on one 512-byte disk sector—a sector being the smallest block of data DOS disk operations will transfer. Even if your application stores and accesses data or records in smaller chunks, there's no such thing as a partial sector read. Everything that's in a sector gets sucked up into memory until whatever part or parts your application needs are selected. Writes are then essentially the same process in reverse.

Buffering is critical to the efficient operation of your computer. Too few, and your computer often has to wait until it has a buffer free before it reads or writes more data—and when *it* waits, *you* wait. The recommended number of buffers depends on your system, and the guidelines shown in Table 20-1 seem reasonable with most hard disk systems. (Systems with floppies only generally require fewer buffers.)

If you're using many subdirectories, Microsoft recommends that you roughly double the number of buffers—to 20, for example—using a total of up to nearly 16K of RAM. If you've installed DOS 5.x in the high-memory area (HMA) by specifying the DOS=HIGH option in the CONFIG.SYS, the BUFFERS will also be located in high memory—and that means you aren't taking all that memory away from your applications.

Table 20-1 The number of BUFFERS recommended by Microsoft, based strictly on amount of RAM installed.

Key combination	Action
Left arrow	Moves the cursor back one character
Right arrow	Moves the cursor ahead one character
Ctrl-left arrow	Moves the cursor back one word
Ctrl-right arrow	Moves the cursor head one word
Home	Moves cursor to beginning of line
End	Moves cursor to end of line
Esc	Clears the command currently displayed

There are different kinds of buffering, however. DOS buffers represent one kind, and disk caching another. These two types, while quite different, perform much the same function—although even with caching some buffers are usually beneficial. Unless you're really cramped for memory, you'll do better using disk caching than relying on DOS buffers. I like five buffers as a compromise with a system that has caching. Even with a lot of subdirectories.

Unfortunately, many applications programs—only able to read the BUFFERS= statement in the CONFIG.SYS and unaware of SMARTDRV or any other disk caching you might be using—will try to bully you into increasing the number of buffers. The installation programs for some go so far as arbitrarily doing it whether you want them to or not. If you're using disk caching and that happens, change the BUFFERS= statement back to read the number or buffers that you think you need, because excess buffers not only waste memory but actually slow your system down.

Although poorly documented and described only as a *secondary buffer cache*, the second parameter indicated above (*mm*) appears to be a continuation of the so-called "look ahead" buffer scheme introduced in version 4.x. It's intended to benefit applications that perform sequential rather than random disk reads/writes (word processors, compilers, and spreadsheets typically do sequential read/writes, large databases usually don't).

The magic number for these secondary buffers seems to be about 8, although even Microsoft advises against any when using SMARTDRV.SYS—which you can interpret as meaning *any* disk cache. In any case, buffers, while important, should not be treated lightly. And even if you don't remember anything else about them, just remember that, when it comes to BUFFERS=, more is definitely not better.

CALL

Internal to COMMAND.COM

Function Transfers temporary control from one batch file to another without terminating the parent file.

Usage In batch files only.

Origin Version 3.3 and later.

Format CALL [*d:*][*path*]*filename*[*argument*]

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Not supported.

One of the few batch file enhancements DOS has added since the early days, CALL is surely one of the most powerful tools available to batch file programmers. After executing the called file, control is returned to the parent program rather than just stopping there.

One of the things I like best about it is that CALL lets me do modular batch files. Aside from the fact that sometimes the same "module" is suitable for use with several applications, I can break things into smaller packets that are easier to develop and debug—and keep my main batch neater and cleaner in the bargain.

Here, for example, are two consecutive CALL lines pulled from one of my batch files—a batch file that itself is called by a conditional IF statement in another batch file:

```
call addpath c:\dos\4dos
call \dos\batch\old_date
```

The first line calls ADDPATH.BAT, which adds something to my path (which I normally keep as small as possible to keep the size of the environment down to under DOS's old 160-byte default size) when I open a particular window in a multitasking environment—now increased to 256 bytes. This is a little module that I use a lot.

The next CALL is for OLD_DATE.BAT, which checks the date to see if it's time to unfragment my hard disk and does it if it's time. Using CALL keeps the dozen lines or so of OLD_DATE.BAT from cluttering the calling file.

CALL is definitely one of DOS's better efforts; certainly one of the most useful tools in its batch file bag of tricks.

CHCP

Internal

Function Displays or changes the active code page setting for all devices that support code page switching.

Origin Version 3.3 and later.

Format CHCP *nnn* (to change to different code page)
or CHCP (to display the current setting)

Switches None.

Errorlevel Not supported.

Network use Supported.

Most of you reading this book will never need this one. CHCP is one of several interrelated functions DOS has added as it has become more multilingual. This one has to do with setting hardware—displays, printers, etc.—to deal with other languages.

We all use code page, however, or at least the one DOS gives us by default. If you don't believe me, just try typing CHCP and Enter on the command line sometime. Most likely it will return:

```
Active code page: 437
```

437 is the page code for the United States. Other code pages supported by 5.0 are:

850	A rather generic multilingual setting
860	Portuguese
861	Icelandic
863	Canadian-French
865	Nordic

This isn't the same as COUNTRY or KEYB. Nor is it even a stand-alone function. Rather, it's just one of several interrelated utilities. While you can use CHCP at any time to check the current code page as above, before even attempting to change to a different code page, you must load NLSFUNC. And before that you must prepare that code page with MODE. Otherwise CHCP will return the message:

```
Bad function "nnn"
```

Changing over to a different code page isn't something any user is likely to have to do very often, and it's something that can, for the most part, be managed by a batch file that contains the necessary commands and parameters.

CHDIR (or CD)

Internal to COMMAND.COM

Function Display the current directory name or changes to a different directory.

Usage Can be executed from the DOS command line or batch files.

Origin Version 2.x and later.

Format CHDIR [*path*] (to change to a different directory)
or CHDIR .. (to return to the parent directory)
or CHDIR (to display the current path)
Note that CD can be substituted for CHDIR in any of the above.

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Supported.

Changing directories is a simple task, yet a confusing one to many DOS users because of some of the subtleties, particularly when and when not to use backslashes. And then those two dots... Let's start with the backslash. If you get that one down cold you can pretty well forget the others.

Any time you follow CD (or CHDIR) with a backslash DOS goes clear back to the root directory to start your new directory path. For instance, I have a TAB directory on drive D:, the drive where I keep all my files. If D: is my default drive it doesn't matter where on drive D: I happen to be at the moment, all I have to do to change to my TAB directory is enter:

```
CD \TAB
```

However, in this case there's nothing in the TAB directory except subdirectories for several books. So suppose at this point I want to go to a third-level subdirectory: \TAB\BEYONDOS\FINISHED. If I'm already in the TAB directory I have two choices. I can do:

```
CD \TAB\BEYONDOS\FINISHED
```

The backslash starts me back at the root directory level again. Or I can just type the rest of the path from where I am, in this case:

```
CD BEYONDOS\FINISHED
```

Note that there's no backslash at the beginning of that last one. If I put one in, like this:

```
CD \BEYONDOS\FINISHED
```

I'll get the following message:

Invalid path "\BEYONDOS\FINISHED"

And it is invalid because there's no BEYONDOS directory at the root level where the backslash took DOS to look for it. So the rules are:

- No initial backslash to go forward with your path.
- Initial backslash only to start back from the root.
- The backslash by itself (CD \) returns you to the root directory.

The two little dots are usually more of a nuisance than they're worth. They take you back to the *parent* directory—not to be confused with the *root* directory. The parent directory is one step back from where you currently are. If you are in a first- level directory, then of course the root directory is the parent. Otherwise, you go just one step back. Looking again at the path I set earlier, TAB\BEYONDOS\FINISHED, the dots would take me back to TAB\BEYONDOS. Which is not usually where I want or need to go.

As far as using just plain old CD or CHDIR to find out where you are—Why? All you have to do is type PROMPT $P and Enter on the command line (or better yet include it in your AUTOEXEC.BAT file) and DOS will always display your current path without you having to ask. More about that when I explain PROMPT in more detail, later.

CHKDSK

External

Function Examines the integrity of the File Allocation Table (FAT), directories, subdirectories, etc., and analyzes the status of system memory.

Usage Can be executed from the DOS command line or batch files.

Origin Version 1.0 and later.

Format CHKDSK [*d:*][*filename*][*.ext*][*switch(es)*]

Switches (other than the drive designation, all CHKDSK switches and options are version 2.0 and later.)

- /? Displays on-screen help including syntax, switches and etc.
- /F Attempts to fix errors on disk. *Caution!* Can cause loss of data.
- /V Verbose mode, lists all filenames, directories, etc.

Errorlevel Not supported.

Network use Not supported.

This command is absolutely essential and should be an everyday part of every DOS user's equipment. In fact, my AUTOEXEC.BAT runs CHKDSK automatically the first time I boot up every day—the last thing I want to do is work with a sick hard disk, which puts not only that day's work but possibly a whole lot more in danger.

Given a healthy typical hard disk, CHKDSK will normally report something like what's in Fig. 20-2. You can see there's a lot of information presented. The top grouping analyzes the disk itself and how it's being used. The second group-

```
Volume Serial Number is 220C-4456

  33396736 bytes total disk space
    544768 bytes in 20 hidden files
    110592 bytes in 49 directories
  27158528 bytes in 1165 user files
   5582848 bytes available on disk

      2048 bytes in each allocation unit
     16307 total allocation units on disk
      2726 available allocation units on disk

    655328 total bytes memory
    529200 bytes free
```

20-2 The CHKDKS report from a healthy hard disk, with no lost clusters or other defects found.

ing tells you (if you read between the lines) that each allocation unit—often referred to as a cluster—contains four sectors (2048/512=4). And the last two lines have nothing to do with disks, but rather how much conventional memory is available and how much of that is currently not being used.

If a disk has two or more partitions, each must be addressed separately because CHKDSK doesn't in fact interpret the word *disk* in the same way you might. A disk partition is a *virtual* disk—a disk within a disk—and these virtual disks are disks as far as CHKDSK is concerned.

Of course, things don't always go smoothly. Even if you have no reason to suspect you have a problem, CHKDSK might find one, which is why it pays to run it routinely on some kind of regular schedule. With any luck, you'll discover problems before they really *are* problems, as in the following example—here a floppy I had downloaded files to work on with my laptop:

```
Volume MOSTLY_DOS1 created 02-26-1990 12:36p
Errors found, F parameter not specified
Corrections will not be written to disk

B:\BEGINNIN.CH

Allocation error, size adjusted

11 lost allocation units found in 2 chains.
Convert lost chains to files (Y/N)?
```

An allocation error! Whatever that means it isn't good. Something is wrong and this is why you always want good backups. Here, I knew the original was good so I just erased the damaged file and copied the original to this disk—but not until I'd captured the 11 lost allocation units found in the two chains CHKDSK also reported on the disk by running CHKDSK again, but with the /f switch.

Consider what might have happened if I hadn't discovered the problem when I did. I might have spent hours working on a corrupted file, then copied it back over the good file on my hard disk. I also might have lost the whole works; in this case, as the filename suggests, the beginning chapter of a book—a nearly finished chapter because I never put extensions on text files until then.

Don't let the *Convert lost chains to files? (Y/N)* prompt intimidate you. True, CHKDSK isn't going to let you move on until you tell it yes or no. But unless you've used the /f switch, it doesn't mean a thing and I wish DOS would change the message to something like *Run CHKDSK /f to convert lost chains to files? (Y/N)* instead. Unless you've used the /f switch, CHKDSK can do nothing but report. No matter which way you answer, it can neither help or make things worse at this point—except possibly scare the Hell out of you.

Of course, problems like this rarely just go away, and sooner or later you have to either bite the bullet or just leave all that space in limbo—DOS can't reuse the space until you do something. This is where the /f switch comes in. Don't use this

one carelessly, however. CHKDSK /f can be as dangerous as it is important because you can't undo it once it's done. Any and all recovered clusters and chains are instantly renamed as FILE000*n*. So don't ever just tag it on unless you not only know you've got a problem but know for sure what you want to do about it.

One of the CHKDSK's lesser known virtues is what it will tell you if you give it more than just a disk or drive to check. If you give it a specific path and file (or wildcard group of files) it will tell you something interesting. I asked it to check D:\TAB\DOS5*.* and it reported:

```
D:\TAB\DOS5\ONCEOVER.BAK Contains 3 non-contiguous blocks
D:\TAB\DOS5\ONCEOVER.SEC Contains 11 non-contiguous blocks
D:\TAB\DOS5\ONCEOVER.SE2 Contains 2 non-contiguous blocks
```

Loosely translated, those are *fragmented* files—and I had unfragmented my entire disk just an hour or so earlier and I knew it was no big deal. You might want to check some of the files on your hard disk that way, and if they are too fragmented, give some thought to unfragmenting your disk with a disk utility program.

Because CHKDSK is so important, in case you have a problem it's not a bad idea to keep a copy handy on a floppy disk—preferably a bootable floppy. There are some problems, sometimes relatively minor, that can send CHKDSK off into limbo if you run it from the same disk that has the problem. It's a rare occurrence, but it can happen and you shouldn't panic.

The /v (verbose) switch names every file in every subdirectory. Unlike the /f switch, however, it's quite harmless. I can't really think of much practical use for this switch. I never could—and there is even less need now with DOS's newly enhanced DIR command. Anything /v can do, it can do better. But otherwise, I think you can forget this switch.

COMMAND

External (.COM)

Function Loads a new or secondary command processor.

Usage Can be executed from the DOS command line.

Origin Version 2.x and later.

Format COMMAND [*d:*][*path*][*switch(es)*]
or SHELL = [*d:*][*path*] COMMAND.COM [*switch(es)*] (when invoked from the CONFIG.SYS)

Switches (the current list no longer includes /msg, a switch valid only with floppy-disk systems but supported by 4.x)

/? Displays on-screen help including syntax, switches and etc.

ctty-dev Specifies different device for command input and output (see CTTY command for more information).

/e:*nnnn* Specifies the size of the environment in bytes (the default is 160).

/p Makes the new copy of the command processor permanent.

/c *string* Tells the command processor to perform the command specified in the string, then exit.

Errorlevel Not supported.

Network use Not supported.

Among DOS's more esoteric commands are all the functions of COMMAND .COM. COMMAND.COM's primary function is to provide an interface between the user and DOS's hidden files (the kernel and BIOS). To accomplish this, a copy of COMMAND.COM (or some substitute acceptable to DOS) is loaded automatically every time you start your system. Here you see it as it appears when you run SET—ensconced as the COMSPEC, the command interpreter specified in the environment:

```
COMSPEC = C: \ COMMAND.COM
TODAY = 10-18-1990
PROMPT = $P$_
PATH = C: \ DOS; \ C: \ DOS \ BATCH;C: \ DOS \ UTILITY
```

Everything you ask DOS to do is communicated through the command interpreter, and many of the error messages DOS reports are generated by it. In addition, commands listed in this section as being Internal are actually functions of COM MAND.COM and are executed directly by it. External commands, on the other hand, are executed *through* COMMAND.COM—they are actually performed by

stand-alone programs external to COMMAND.COM. These external commands include all those other .COM and .EXE files in your DOS directory.

To this extent, COMMAND.COM is just a workhorse; not very pretty, but essential and, more important, invisible. At this point, you never have to even give it a thought. To format a disk, for instance, even though you have to go through COMMAND.COM to get to FORMAT.COM, you don't have to pass FORMAT as a parameter with something like:

```
COMMAND FORMAT /...
```

DOS loads it automatically. Programs you run load secondary copies automatically.

COMMAND.COM is a peculiar program in several respects, in part dating back DOS's early days when even 64K was a lot of memory. It's actually divided into two parts. Most people know that COMMAND.COM (or something like it) has to be in place for DOS to function—but not *all* of it. Not when you're running other programs. So COMMAND.COM is written in a way that lets other programs overwrite the parts it won't need while they're running, reloading those parts from disk when a program that has overwritten them is terminated so you have the whole thing back again.

The environment (/e) variable can be set only from CONFIG.SYS and is necessary only if you can't live within the 256 bytes DOS gives by default. In my opinion there are better ways to use memory than waste it on a fat environment in most cases. By keeping the path pared down to a decent length (which runs faster searches anyway) you can keep your environment down to a reasonable size.

Another switch that should only be invoked when specifying COMMAND .COM as the primary command processor from the CONFIG.SYS is /p, which makes that COMMAND.DOM permanent as far as the work session is concerned. If the /p switch is appended to COMMAND.COM when you invoke it from the command line, it makes that secondary copy permanent, which then means you can't exit back to the primary command processor, leaving you no way out except by rebooting.

The /c *string* switch is rarely used. Prior to version 3.3—before CALL was introduced—this option was used instead of the CALL command by experienced DOS users. By executing a batch file through COMMAND.COM, DOS loads a secondary copy of the command processor, executes the command specified—in this case a batch file—and, when finished, exits back to the realm of the primary command processor, i.e., the next line in the original batch file. Of course, loading a secondary command processor takes time and, by comparison, CALL typically does the same job as much as three times faster than the old COMMAND /c method.

COMP

External (.EXE)
Enhanced

Function Compares contents of two files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0 and later.

Format COMP [*d:*][*path*][*filename1*][*.ext*][*d:*][*path*][*filename2*][*.ext*]

Switches

/? Displays on-screen help, including syntax, switches, etc.
/a Displays differences in ASCII format.
/c Disregards case of ASCII letters when comparing files.
/d Displays differences in decimal format.
/l Displays line numbers for differences.
/n:*nnn* Compares only the first *nnn* lines of each file.

Errorlevel Not supported.

Network use Supported.

As you can see, COMP has been enhanced by the addition of several switches. For my money, however, I'd rather they had put the effort elsewhere. As with many of the DOS utilities that trace their origins to the earliest days, even in its enhanced state COMP has been supplanted by more powerful utilities and represents the kind of thing DOS critics point out as being outdated.

COMP is really pretty crude. If the files you want compared are of different sizes—even by a single byte—COMP doesn't bother to check anything else. It just reports:

```
Comparing E:TEST and E:TEST2...
Files are different sizes
Compare more files (Y/N) ?
```

It doesn't take a genius—or a special program—to figure that out. Even if the files are, byte for byte, the same exact length, COMP really doesn't tell you that much. Here, I doctored one of two identical files, but in a way that made the total bytes equal again, and got:

```
Comparing E:TEST and E:TEST2...
Compare error at OFFSET 415
file1 = 20
file2 = 78
Compare error at OFFSET 416
```

```
file1 = 20
file2 = 78
Compare error at OFFSET 417
file1 = 20
file2 = 78
Compare error at OFFSET 418
file1 = 20
file2 = 78
Compare more files (Y/N) ?
```

Still not much better because it doesn't really tell you anything. The numbers 20 and 78 are hexidecimal, COMP's default style. By adding the /a, you can have the information displayed in ASCII characters, which will tell you that four consecutive blank spaces have been replaced with x's.

If you really want to know if one of these two files has been corrupted—and if so, how—try DOS's other file comparison utility, called FC. It will show you, for the same two files, not only the actual differences but give you those differences in their proper context.

COMP is admittedly crude. If it finds and displays ten mismatches, the command terminates—though, for most practical purposes, that's certainly enough to establish that data has been corrupted. And that in itself is all most users really need to know when there is a question of integrity between two copies of the same file.

FC, in comparison, won't just turn up its nose because two files are different lengths. It assumes you're smart enough to figure that out for yourself, and so do I. That's why, despite the new enhancements, I think you might want to erase COMP from your hard disk like I did.

COPY

Internal to COMMAND.COM

Function Copies one or more files to a different location or media. COPY can also append files.

Usage Can be executed from the DOS command line or batch files.

Origin Version 1.x.

Format COPY [*d:*][*path*][*file.ext*] [*d:*][*path*][*file.ext*] [*switch(s)*]
or COPY [*d:*][*path*][*file.ext*] + [*d:*][*path*][*file.ext*] [*d:*][*path*][*file.ext*]
or COPY [*d:*][*path*][*file.ext*] [*device*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/a Copies ASCII text files—not normally required and, when appending files, it is the default.

/b The default except when appending files. Copies binary files—tells the command processor to read the number of bytes specified by the file size in the directory.

/v Verifies that the copy process has been completed satisfactorily. It doesn't verify the integrity of the data copied.

Errorlevel Not supported.

Network use Supported.

Little has ever been—or will be—done to enhance COPY much beyond its current state. It's adequate for most purposes, however, and we'd better be content with it considering that XCOPY, one of DOS's much more powerful external file copy utilities, is 18,213 bytes long all by itself in release 5.0, along side of COMMAND.COM, which weighs in at a comparatively slim 41,765 bytes.

In addition to its traditional role of copying files from one device to another, COPY can also be used to copy the output of one device to another. One of the most useful applications of this is in creating simple one- or two-line batch files like the one below (making sure to put a Return at the end of each line except the last):

```
COPY CON C:\DOS\BATCH\ATTACH.BAT
@C:\DOS\SQUISH\ATTACH %1 %2
^Z (accomplished by pressing the F6 key)
```

Line one simply says that you want to copy the output of the console—the display device—to a disk file called ATTACH.BAT and the directory you want it in. Line 2 executes a program by the same name—ATTACH in this case—that's in a direc-

tory that isn't part of the path, passing parameters to it as if it was in the path or on the default directory. Doing it this way, if the directory the program is in happens to be the default, the command works just the same. This, a favorite trick of mine, is one of the ways I keep my PATH statement short. Of course, with 5.0 you now have the option of doing this sort of thing through DOSKEY with macros. COPY CON is still a good trick, however, and works with any version of DOS.

It's rare that you'll ever have to use any of COPY's three switches. There are, of course, exceptions. In some cases it might be necessary, when appending a group of files into a single larger file, to use the /b switch. For instance, I sometimes have occasion to create compressed files that are self-extracting using PKWARE utilities which, because the process involves several steps, I manage with the following batch file:

```
PKARC A %1 %2 %3 %4 %5 %6 %7 %8 %9
COPY/B PKSFX.PGM + %1.ARC %1.EXE
```

After compressing one or more individual files and appending them into a single file, that file gets appended to the program that makes the package self extracting—and there's /b because at least one if not both of the files getting copied into the .EXE file are binary.

In most cases the defaults are fine, although few people who understand what /v *doesn't* do want it slowing down their systems—which is about all it *does* do.

The /v switch is certainly one of the most misunderstood features in DOS. It does *not* test the data copied against the original. It merely verifies that no hardware errors have occurred. If the data has been corrupted in the process, tough. Not that this happens very often—certainly not unless your computer is coming down with something—but it does happen and the verify switch won't tell you. If you have to verify the actual data, use FC to compare the files—or COMP if you haven't followed my advice to erase it.

For all that DOS has matured down through the years, COPY is just one of those rudimentary throwbacks. Still, it does the job—I use it probably ten times for every time I use XCOPY or REPLACE because it's quick and easy. Which is really what an operating system is supposed to be now, isn't it?

COUNTRY.SYS

Device driver

Function Enables DOS to use international time and date conventions, currency, case conversions, decimal separators, etc.

Usage Can be loaded only from the CONFIG.SYS.

Origin Version 3.x and later.

Format COUNTRY = *xxx*[*yyy*][*d:*][*path*][*filename.ext*] (where *xxx* specifies the country code, and *yyy* specifies the code page for the country)

Switches None.

Errorlevel Not supported.

Network use Not supported.

This is one of a group of utilities geared to multilingual usage, and must users reading this book can ignore it completely. DOS sold in the U.S. defaults to the conventions used in the United States. Country codes affect four MS-DOS commands: DATE, BACKUP, RESTORE, and TIME.

COUNTRY.SYS can be installed only from the CONFIG.SYS at the time the system is booted, although code pages can be changed from the command line in mid-session provided they've been properly prepared.

The documentation supplied with DOS for COUNTRY.SYS is extensive, consisting mainly of tables of Nationality, Code pages, Country codes, and Time and Date formats. Because this information is lengthy, the number of users who require it small, and using it at most simply a matter of looking up the code numbers, those who do have occasion to use COUNTRY.SYS are referred to the Microsoft or OEM documentation.

CTTY

Internal

Function Changes the terminal device used to control your system.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.x.

Format CTTY *device*

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Supported.

This is another of MS-DOS's more obscure commands, probably of greater importance in the days of its version 2.x origins—at least at the DOS level where the user is directly involved. A number of popular programs actually do this sort of thing routinely, including programs that allow multi-user access and file transfer programs for downloading from laptops to desktops, but these kind of programs generally do all the housekeeping and deal directly with the DOS kernel, invisible to you.

If the serial port is used to access the device to be used by CTTY, it must be initialized by MODE for baud rate, parity, bits, and stop bit before executing CTTY. In the case of programs like those cited above, they do it for you.

If you like to play games with obscure commands and usages, however, you can use CTTY to suppress normal screen output, say during some lengthy operation, by specifying NUL for the device, then returning it to the normal device when finished, as shown below in DUMB.BAT:

```
ctty nul
echo HELLO
ctty con
```

HELLO, of course, (or anything else that happens between the two CTTY commands) never gets echoed to the screen. This gets rid of everything. Even error messages that might be important—which could leave you hanging staring at a blank screen, unable to even get control again without rebooting. Well, its name *is* DUMB. And if it's just screen output you want to suppress, there's a better way. Just redirect it to NUL (>NUL) as I did with CHKDSK in the example below.

```
B:\FILES\DOS\XYWRITE
    Invalid sub-directory entryB:\FILES\DOS\DV
    Invalid sub-directory entryB:\FILES\DOS\SUPRCALC
```

```
Invalid sub-directory entryB:\FILES\TAB\DOS5
Invalid sub-directory entryB:\FILES\SUPRCALC
Invalid sub-directory entry
```

The display, as you can see, reached the screen slightly scrambled, but the message came through loud and clear—this is a disk with serious problems. Not that you would ever want to run CHKDSK this way. This just happened to be one where I knew I'd get an error message to illustrate the point.

CTTY is there and in special circumstances serves a need. But if there's another way I'd think twice before I went this route.

DATE

Internal to COMMAND.COM

Function Displays the date or allows you to change the date.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.x.

Format DATE (displays the date as currently set)
or DATE [*mm-dd-yy*] (sets or resets date, month, day and year)
or DATE [*mm/dd/yy*] (alternate entry mode)
Note that the date format will vary if code pages other than the default United States code page is used.

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Supported.

If DOS finds no AUTOEXEC.BAT file when you boot your machine, the system will pause and prompt for both DATE and TIME, regardless of whether your hardware includes full calendar support or not. The presence of *any* AUTOEXEC.BAT in the root directory defeats this feature, however—even if it's nothing but an empty file.

In most cases, if your machine contains an internal clock/calendar, the DATE command will set or reset the date portion, although the clocks provided by many expansion boards generally must be set through some separate proprietary utility. This is relatively uncommon today because most newer machines have clocks built into them. The DOS DATE command can generally read the date even from clocks it can't reset, and even in those cases can reset the machine to override the installed clock and reflect whatever date is set through it for the duration of that work session.

With a little imagination, you can use the DATE function for any number of useful things beyond just dating files as you create or modify them. One of my pet tricks uses the date as a variable in the environment, easily accomplished from my AUTOEXEC.BAT with the command:

```
ECHO|MORE|DATE > DATE.BAT
DATE
```

The first of those two lines runs the DATE command and redirects the output to a batch file (the ECHO|MORE business just provides the necessary carriage return at the end so you can do this from a batch file—I do it from my AUTOEXEC

.BAT). The next line runs the batch file then, returning something like:

```
Current date is Tue 02-26-91
Enter new date (mm-dd-yy)
```

Don't worry about the second line. The word *Current* gets DOS's attention because it knows there's another batch file, CURRENT.BAT, on the default directory, so it runs CURRENT. CURRENT then takes the date portion (02-26-81) which is the fourth character string after the word *Current* and passes it as a parameter (%4) for SET TODAY= to the file CURRENT.BAT, which simply reads:

```
SET TODAY=%4
```

And when you check your environment with a SET command, there it is:

```
COMSPEC=C:\COMMAND.COM
PATH=C:\DOS;C:\BATCH;C:\UTILITY
PROMPT $P$_
TODAY=02-26-91
```

Okay, so what? Well, check that against the date you recorded the last time the system was booted by running another batch file I call OLD-DATE.BAT. OLD_DATE.BAT is just a copy of the file that was my DATE.BAT the last time I booted, and it runs a batch file called CURRENT.BAT, which compares the old date with the current date in the environment.

```
if !%TODAY%==!%4 goto END
```

If the two are the same, it skips to the label :END, otherwise it reads and executes the next line, which in my system unfragments my hard disk—something I do once a day religiously—and runs a couple of other once-a-day housekeeping jobs.

Note that I have more than one CURRENT.BAT file, and that's fine as long as they're in separate directories and preferably directories that aren't in the path—which is one of the reasons that setting the PATH is one of the last things I do in my AUTOEXEC.BAT. I just have to make sure I've made the directory for the CURRENT.BAT file I want to be the default directory.

This trick is by no means limited to just the date itself. If I passed the third string—TUE in the above example—as a parameter to SET in the environment, I could use it to do something every time the day did or didn't match, even if only to echo something like "Call your mother."

MS-DOS is programmed to handle months and years correctly, regardless of whether the month has 28, 29, 30 or 31 days—but only until the year 2079, so work fast. And don't forget to call your mother.

DEBUG

External (.EXE)

Function Starts DEBUG, a comprehensive program that allows you to examine data and create, edit, and test executable files.

Usage Generally executed from the DOS command line.

Origin Version 1.x.

Format DEBUG [*d:*][*path*][*filename.ext* [*argument_list*]]

Switches

/? Displays basic on-screen help.

Note that the following list of commands are available by entering a question mark and the appropriate letter(s) at the DEBUG prompt, after loading DEBUG:

a Assembles 8086/8087/8088 mnemonics.
c Compares two areas of memory.
d Displays the contents of a memory area.
e Enters data into memory starting at specified address.
f Fills a range of memory with specified values.
g Executes a program currently loaded into memory.
h Performs hexidecimal arithmetic.
i Displays one byte from a specified port.
L Loads a file or file sectors into memory.
m Moves a block of memory.
n Assigns filenames for the L and w commands and assigns parameters to a file being tested.
o Sends a byte to an output port.
p Executes a loop, a repeated string instruction, a software interrupt, or a subroutine call to completion.
q Ends a DEBUG session.
r Displays the contents of one or more registers.
s Searches a portion of memory for specified byte values.
t Executes one instruction and displays the contents of all registers, flags, and the decoded instruction.
u Unassembles bytes and displays the corresponding source statements.
w Writes the file being tested, edited, or created to a disk.
xa Allocates expanded memory.
xd Deallocates expanded memory.
xm Maps expanded memory pages.
xs Displays the status of expanded memory.

Errorlevel Not supported.

Network use Not supported.

Debug is more than just a command. It's a working environment that carries with it its own whole subset of commands. DEBUG is surely one of DOS's more powerful tools, but one not recommended for beginners. I would venture, however, that many people who should be using it aren't. You shouldn't let DEBUG intimidate you. Because of all it can do, I've devoted one whole chapter to it. All I've done here is list its various commands for reference.

Those familiar with DEBUG will notice several commands—those two letter jobs that start with x's—that weren't there through version 3.3. They were among the few new features anywhere in 4.x. Version 5.0 adds nothing new to DEBUG beyond that, however.

DEL (ERASE)

Internal to COMMAND.COM

Function Deletes specified file or files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.x.

Format DEL [*d:*][*path*][*filename*][*.ext*][*/p*]
or ERASE [*d:*][*path*][*filename*][*.ext*][*/p*]

Switches

/? Displays on-screen help including syntax, switches and etc.

/p Causes DOS to pause and prompt for Y or N before deleting specified file or each file by name if wildcards have been used.

Errorlevel Not supported.

Network use Supported.

This is one of DOS's most straightforward commands. It's also one of DOS's laziest, not even doing half the job—which is a very fortunate thing for you. All DOS actually does when it "erases" a file is change the first character of the directory listing to a peculiar ASCII character no one would ever use in a filename, and releases whatever disk space the file occupies so DOS can reuse it when it needs more space to write something else.

Shown in Fig. 20-3, a peculiar character, an ASCII 229, replaces the original first letter—here, in both cases, an E. The last number is simply the number of the first allocation unit used by that file.

```
BORROWED          150   10-13-90    2:09 pm       2
CATBLURB         2361   10-10-90    2:56 pm       3
σDIT              326    9-08-90   10:52 am       6
σXTENDED        14869   10-11-90    5:04 pm       7
```

20-3 An excerpt from the actual root directory of a disk, showing two "erased" files. This is the data UNDELETE uses to locate and recover erased files when the MIRROR undelete tracker hasn't been used.

ERASE does not delete or in any way corrupt the data itself. However, when DOS sees the peculiar character it has substituted for the first character in the filename, it knows the file has been erased and will therefore ignore the file when you call for an ordinary directory listing.

Until 5.0, while this fact was well known and used as the basis of a number of 3rd-party unerase utilities, Microsoft failed to exploit this fact and DOS offered

no means of undeleting files. In 5.0, you have UNERASE, a most welcome new addition. Note though that UNERASE works only if the filespace has not yet been overwritten with one or more other files. Still, it's nice to know you've got a second chance now with DOS 5.0.

DELOLDOS

External (.EXE)

NEW!

Function Allows you to delete old DOS files from a previous version after successful installation of 5.0.

Usage Used only once following any successful installation from the DOS 5.0 distribution disks.

Origin Version 5.0

Format DELOLDOS

Switches

/? Displays on-screen help including syntax, switches and etc.
/b For use with monochrome instead of color displays.

Errorlevel Not supported.

Network use Not supported.

Note that DELOLDOS is supplied with the upgrade package only because it would serve no use with an original installation of DOS 5.0.

This is a command you run only once—you can't run it twice because it isn't there to run (part of what it cleans up is itself). At least it's gone until the next time you install DOS from your original distribution disks by means of the new DOS 5.0 SETUP program, which replaces the SELECT program supplied with several earlier releases.

One of SETUP's functions is to save whatever DOS version you're currently running under in such a way that should there be a problem DOS 5.0 can be uninstalled and your previous system be quickly and painlessly restored. In addition to requiring one or more floppy disks, this can also require up to a megabyte or so of additional hard disk space, depending on which version you're updating from, until you're sure your 5.0 installation is satisfactory.

```
┌──────────────────────────Warning──────────────────────────┐
│                                                           │
│  Running Deloldos removes all old DOS files from your system,  │
│      making it impossible to recover your previous DOS.   │
│                                                           │
│                                                           │
│          To continue with Deloldos, press Y.              │
│             To exit, press any other key.                 │
│                                                           │
└───────────────────────────────────────────────────────────┘
```

20-4 The warning screen for the DELOLDOS command.

The sole function of DELOLDOS, then, is to remove all your old DOS files once you're satisfied with version 5.0. As is characteristic of the new DOS, DELOLDOS gives you ample warning before doing anything irreversible with the warning screen shown in Fig. 20-4.

It's ironic that we had to wait for this new version of DOS with its many new features and generally forgiving ways before DOS offered such a ready "back door."

DEVICE

Internal to IO.SYS

Function Loads the specified device driver into memory.

Usage Can be used only in the CONFIG.SYS.

Origin Version 2.x.

Format DEVICE = [*d:*][*path*][*filename.ext*[*arguments*]

Switches None.

Errorlevel Not supported.

Network use Not supported.

One of DOS's set-it-and-forget-it commands, DEVICE= is used to load a specified device driver from the CONFIG.SYS. A typical CONFIG.SYS file might contain several DEVICE= commands, but, as shown below, each one can load only a single device.

```
DEVICE = DOS \ HIMEM.SYS
DEVICE = DOS \ EMM386.SYS 2048
DEVICE = DOS \ RAMDRIVE.SYS ....
DEVICE = DOS \ SMARTDRV.SYS ....
LASTDRIVE = Z
FILES = 20
buffers = 5
STACKS = 0,0
DOS = HIGH
```

Notice that, when loading device drivers, the file extension (.SYS) must be included. Note, too, that there is a hierarchy. Until HIMEM.SYS is loaded (286 and higher only), DOS doesn't recognize any memory beyond 640K, no matter how much is actually installed. With HIMEM.SYS loaded, you can use *extended* memory—and you must have extended memory before EMM386.SYS can emulate *expanded memory*. RAMDRIVE.SYS and SMARTDRV.SYS can use extended or expanded memory only if they find it when you try to load them.

Caution: MS-DOS automatically loads the files COUNTRY.SYS and KEYBOARD.SYS when necessary. While these files have the .SYS file extension, they are *not* device drivers. Don't attempt to load either of these with the DEVICE= command, as it will "hang" the system.

DEVICEHIGH

Internal to IO.SYS
NEW!

Function Loads specified device driver into reserved (upper) memory.

Usage Can be used only in the CONFIG.SYS.

Origin Version 5.0.

Format DEVICEHIGH = *d:*][*path*]*filename*.SYS
or DEVICEHIGH [*size*] = [*hexsize*][*d:*][*path*]*filename*.SYS

Switches

size = *hexsize*

Errorlevel Not supported.

Network use Not supported.

On 386 and higher systems you can use DEVICEHIGH= instead of the traditional DEVICE= in the CONFIG.SYS to load device drivers into expanded memory, and map them to valid, otherwise unused DOS address space above 640K. This presupposes that you have HIMEM.SYS loaded, and that you have EMM386.SYS loaded to emulate expanded memory that can be mapped to empty address space above 640K. Any DEVICEHIGH commands must come after these prerequisites are established.

On many 386 and higher systems, as much as 128K or more of usable space can often be recovered above 640K—in addition to the 64K page frame area if you're using expanded memory and the HMA where you can load much of the DOS kernel with the DOS=HIGH command. While this upper memory is usually fragmented into smaller blocks, unlike conventional memory, it's possible in many cases to use most of it.

Determining which drivers can be loaded into upper memory blocks using this function can be either hard or easy. If you have only one or two you'd like to try to load above 640K, little or no experimentation might be required. Dumb luck, however, will often serve for only one or two. The more device drivers you try to load, the more difficult the task.

The size=*hexsize* switch is used to specify some minimum amount of memory that must be available in a contiguous upper memory block before DEVICE HIGH will attempt to load the specified device driver into reserved memory. It's not required except in the case of certain device drivers that might actually require a larger block of memory to run in than might be apparent at the time they load.

That's the good news. The bad news is that you'll usually find out that you need to use this one only when you have a problem. See the chapter on memory management for a more detailed discussion.

DIR

Internal to COMMAND.COM

Function Displays a list of files in a directory or directories.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0, all releases.

Format DIR [*d:*][*path*][*name/wildcards*][*switch(es)*]

Note that switches can be combined as needed. If more than one switch is used, however, don't separate with spaces.

Switches

/? Displays on-screen help including syntax, switches, etc.

The following are directory listings by attribute:

/a Displays all files and directories, regardless of attribute.
/a:a File ready for archiving (backup).
/a:d Directories only (no files).
/a:h Hidden files.
/a:r Read only.
/a:s System files.

The following are directory-sort options:

/o:n A–Z alphabetically (the default).
/o:e A–Z by extension.
/o:s Smallest first.
/o:d Oldest first.
/o:g directories before files.

Note that, with the above two types of directory options, inserting a minus sign (–) immediately following the colon in any of the above reverses the meaning. The switch /a:–d, for example, displays files but not directories.

The following are miscellaneous switches:

/b Lists FILENAME.EXT only, and no other information.
/l Displays file and directory names in lowercase.
/s Shows every occurrence, including all subdirectories.

And the following switches are also still supported:

/p Pauses scrolling after each screen page. DIR determines page size by calling ANSI.SYS if loaded, and defaults to a 24-line display if not loaded.
/w Wide display, with as many as five filenames per line.

Errorlevel Not supported.

Network use Supported.

No other single command has been enhanced more in 5.0 than the lowly DIR command. Those familiar with earlier versions will find it still works the same as always with the limited repertoire of options previously available.

One of the most interesting and useful of the newly added options is /s, which allows you to search an entire disk for a misplaced file. Type DIR *.SGT/S at the DOS prompt and you'll see something like that shown in Fig. 20-5.

```
 Volume in drive C has no label
 Volume Serial Number is 220C-4456

 Directory of C:\DOS\XYWRITE

FORMAT   SGT      5712 03-03-90   3:46p

1 file(s)          5712 bytes
                5013504 bytes free
```

20-5 One of the most important new switches added to the DIR command is /s, which has been used here to search an entire hard disk for a particular file.

Combined with other switches, DIR can now conduct wildcard searches for not only individual files, as in the above, but also groups of files, reporting any matches in any and all subdirectories.

To make life easier, version 5.0 even provides an easy way to make whatever combination of DIR switches suits your purposes your personal default setting so you don't have to do the extra typing every time. Microsoft has done it by adding a new internal command called DIRCMD for making one or more of these switches a part of your environment. The syntax for the command (which can be included in your AUTOEXEC.BAT) is:

SET DIRCMD = *switch(es)*

Note that only switches follow the equal sign. The DIR part is taken care of by DIRCMD, so yours might read something like this:

SET DIRCMD = /o:n/a: – h/p

Remember that when multiple switches are used, as in the above example, do *not* insert spaces between them. You can precede a switch or list of switches with a single space, but that's optional.

As with any environment variable, DIRCMD can be reset or altered at any time and as often as desired—or removed entirely by entering SET DIRCMN= with no switches specified. However, when desired, you can override switches set with DIRCMD from the command line by including any you want to override on

the command line but changing the sign—adding a minus sign if none was specified or leaving it out if specified in the environment.

While for the most part the new switches are self explanatory and their usage fairly obvious, there's one sleeper in the bunch: /b. It displays only filenames, and nothing else. In final form, the chapters for this book had extensions like .CH1, CH2 and etc. So I issued the following DIR command:

```
DIR TAB\FIVE_O\*.CH? > FIVE.DOC /b/o:e
```

Using wildcards to narrow the search down to just chapters and combining the /b and /o:e, I sorted and redirected the output to a file. All I had to do then was tell XyWrite, my word processor, to print the list of files out as a document.

There are some peculiarities in the new DIR switches, notably the −h and −s switches, either of which, in most cases, simply displays an ordinary directory. I fail to see much practical use for −r, either. But overall they do represent a quantum leap forward from the seemingly random—and in the case of especially long lists, sometimes incomprehensible—DOS DIR of old.

DIRCMD

Internal to COMMAND.COM
NEW!

Function Puts optional DIR switches in the environment as defaults.
Usage Can be executed from the DOS command line or in batch files.
Origin Version 5.x.
Format SET DIRCMD = *switch(es)*
Switches All DIR switches—see DIR for complete listing.
Errorlevel Not supported.
Network use Supported.

The only new internal command introduced in version 5.0, DIRCMD adds considerably to the flexibility afforded by the many new DIR switches by giving you the option of making whatever combination suits your own personal default setting.

The alphabetical sort option (/o:n) is probably the one that will appeal to most users. To make that your default, you can type the following on the command line (or include it in your AUTOEXEC.BAT so it is done automatically every time you boot):

```
SET DIRCMD=/o:n/p
```

Here I've also added the /p to keep long listings from scrolling off the screen. When using multiple switches, do *not* insert any spaces between them.

Switches passed to the environment as defaults can be overridden by simply typing whatever switch or switches you want on the command line when you want a directory that doesn't match the default profile you've chosen. This doesn't affect the defaults you've established with the SET command. They will remain the defaults until or unless you do one of the following:

- Reboot the system (if you used the SET command from the command line rather than having it in your AUTOEXEC.BAT).
- Run SET DIRCMND= again with a different switch (or switches).
- Run SET DIRCMND= again without specifying any switches to return to DOS's lovable old "take 'em as they come" *default* default.

DISKCOMP

External (.COM)

Function Compares the entire contents of two floppy disks.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0 and later.

Format DISKCOMP [*d:*] [*d:*][*switch(es)*]

Switches (only in version 2.0 and later.)

/? Displays on-screen help including syntax, switches and etc.
/1 Compares first side of diskettes only.
/8 Compares only the first 8 sectors of each track.

Errorlevel

0 Disks compared OK.
1 Disks did not compare.
2 Comparison terminated by user (Esc or Ctrl−Break).
3 Comparison terminated due to hard error (unrecoverable read or write error).
4 Initialization error (not enough memory, invalid drive or command line syntax error, etc.).

Network use Not supported.

This one is really useful only when you're comparing a copy of a disk made with DISKCOPY because it performs an exact track-by-track comparison only. Therefore, a disk created using the COPY or XCOPY commands will in most cases report a mismatch even though the *contents* of all the files might compare perfectly if compared with COMP or FC.

DISKCOMP can't be used in conjunction with SUBST, ASSIGN, or JOIN, with a hard disk, RAM disk, or network disk.

I can't remember the last time I actually used this one. Probably before I learned the difference in the way COPY and DISKCOPY handle files. I put it through its paces this time just so I could say "Okay, it works," and then wipe it off my hard disk.

DISKCOPY

External (.COM)

Function Copies the entire contents of one floppy to another floppy.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0 and later.

Format DISKCOPY [*d:*][*d:*][*switch(es)*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/1 Used to copy single-sided disks (version 2.0 and higher only).

Errorlevel

0 Disk copied successfully

1 Non-fatal read/write error

2 User abort (ESC or Ctrl−Break)

3 Terminated by fatal hard error (unable to read source disk or format target

4 Initialization error (not enough memory, invalid drive or command line syntax error, etc.)

Network use Not supported

With the possible exception of some applications software that, in their documentation, specifies DISKCOPY as the required method of preparing archival backup copies of their distribution disks, DISKCOPY has largely fallen into disfavor with DOS users in recent years. Because it is usable *only* when copying the *entire* contents of one disk to another *identical* disk in an *identical drive*, its use is limited to floppy disks and, for that reason if no other, DISKCOPY generally plays little role in today's hard disk-oriented operating environment.

Traditionally, one of DISKCOPY's biggest drawbacks has been the slavish way it mirrored everything even including the way files on the original disk were fragmented. COPY or XCOPY on the other hand, going file-by-file, do their best to collect the bits and pieces of fragmented files and put them in contiguous sectors, totally unfragmenting when copying to a freshly formatted disk.

Also, DISKCOPY only copies an entire disk to another disk (designated as the *target disk*), reformatting the target as it goes and, of course, wiping out any previous data on the target. Coupled with the fact that DISKCOPY in 4.x never worked the way it was supposed to on this machine, I didn't even keep it on my hard disk.

Still, don't rule this one out just yet. Unlike COPY, DISKCOPY re-creates the tree structure of your source disk as it goes, subdirectories and all. So does

XCOPY with the /s switch—or REPLACE—and, unlike DISKCOPY, you can use them with your hard disk too.

And with DOS 5.0, DISKCOPY is re-emerging as a highly useful program and what to many users are its greatest drawbacks are actually its strongest assets, particularly now that even DOS's legendary, nasty FORMAT has been tamed in this release.

Unlike COPY or XCOPY, DISKCOPY doesn't care what's on a disk. Something or absolutely nothing—it's all the same to DISKCOPY. Try to COPY an empty disk and it will come back:

```
File not found "b:\*.*"
0 file(s) copied
```

DISKCOPY, on the other hand, will make a faithful copy of an empty disk as happily as one that's full.

So, you ask, who'd want to make a copy of an *empty* disk? Well, suppose it really isn't empty but just doesn't have any files left on it. Not files DOS can recognize as files, at least—say after reformatting with with DOS 5's new nondestructive formatter. That empty disk might really have a lot of good, recoverable stuff on it. When I've got a problem, though, I always hate to work on the original if I can help it—if I screw up a copy I can always make another one to play with as long as I've got the original.

With DISKCOPY, I can do exactly that. Working with floppies, you can, believe it or not, DISKCOPY a DOS 5.0 reformatted disk, work with the copy, and still have the original to fall back on if you have to. It's really wild using REFOR MAT to restore all of the lost files from a disk that they were never even on.

Remember, though, that DISKCOPY won't settle for anything but an identical drive (or the same drive) to copy to. I don't have identical drives on any machine in my office any more. Like many of you, I haven't since 3 1/2-inch disks came on the scene and it became almost imperative to have a mix of drive sizes to accommodate whatever came along. Fortunately, you can use the same drive for both the source and target disk. After specifying diskcopy B: B:, you'll see the following prompts:

```
Insert SOURCE diskette in drive B:
Press any key to continue...

Copying 80 tracks
9 Sectors/Track, 2 Side(s)

Insert TARGET diskette in drive B:
Press any key to continue...

Insert SOURCE diskette in drive B:
Press any key to continue...
```

This, as you can see, involves switching your source and target disks back and forth as often as DISKCOPY prompts you to if you have to use the same drive for both. This is a function of how much memory is available and how many bytes you want to DISKCOPY. Basically, it reads as much of the source disk into memory as possible, writes that to the target, reads some more, then writes some more, etc.

When identical drives are available, of course, DISKCOPY works much faster—faster in fact than any of DOS's other copy schemes (except BACKUP) because it doesn't have to bother with amenities like files and all that stuff.

Probably the best way to describe DISKCOPY is crude. Essentially it's a throwback to a time when there were little other than floppies (at best) available as "mass storage" media. But with DOS 5.0, DISKCOPY seems to have a whole new lease on life—I know it's back on my hard disk again.

DISPLAY.SYS

External (device driver)

Function Supports code page switching for the display (console) device.

Usage Can be installed only from the CONFIG.SYS.

Origin Version 3.3.

Format DEVICE = [*d:*][*path*]DRIVER.SYS CON[:] = (*type*[,*code*][,*n*])

The variable *type* specifies the display adapter: MONO, CGA, EGA, or LCD, with EGA supporting both EGA and VGA displays. The variable *code* specifies the country codes supported in this version, which include:

437	United States (default)
850	Multilingual
860	Portugal
863	Canadian-French
865	Norway

The variable *n* specifies the number of additional code pages supported. Values may be from 1 – 12 but are dependent on hardware.

Switches None.

Errorlevel Not supported.

Network use Not supported.

One of the foundation stones of DOS's multilingual support, DISPLAY.SYS must be installed from the CONFIG.SYS if support for code pages other than the default is required. Country code entered with the command specifies the code page that will be in effect initially.

There are some limitations to the ability of DISPLAY.SYS to deal with code page switching. MONO and CGA monitors do not support switching so the value of *n* must be 0. For use with LCD displays the value can be 1, for EGA's it can be 2.

DOS

Internal to IO.SYS

NEW!

Function Tells MS-DOS to maintain a link to the reserved (high) memory area or to load part of itself to the HMA.

Usage Can be used only from the CONFIG.SYS

Origin Version 5.0.

Format DOS=HIGH
or DOS=LOW (default)
or DOS=UMB
or DOS=NOUMB (default)

Switches None.

Errorlevel Not supported.

Network use Not supported.

Installable only from the CONFIG.SYS the DOS command establishes the level at which it will support memory usage by DOS beyond the normal 640K limit for conventional memory. There are two separate issues involved.

The DOS=HIGH command allows DOS to load all but 17K of itself—in excess of 40K—into the high memory area on systems that have extended memory available. This is that mysterious "extra" 64K area DOS can access just beyond one megabyte. Using this option frees in excess of 40K of low (conventional) memory for use by applications programs.

The other option, DOS=UMB, maintains a link to the reserved memory area (640K−1024K) after the CONFIG.SYS had loaded any device drivers via the DEVICEHIGH command. By maintaining this link—which by default terminates when the CONFIG.SYS has run—any upper memory blocks that remain unused are made available for loading TSR's with the LOADHIGH command. This might free up still more conventional memory for use by applications and I recommend it. Both of these options can be included on the same CONFIG.SYS command line provided they are separated by a comma as below.

```
DOS=HIGH,UMB
```

This line, by the way, can be located anyplace in your CONFIG.SYS. Note: the DOS command can also be used in conjunction with 3rd-party memory managers to load a major part of the kernel to the high memory area (HMA).

DOSKEY

External (.COM)
NEW!

Function Recalls DOS commands, allows command line editing, macro creation.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 5.0.

Format DOSKEY [*switch(es)*]
or DOSKEY *macro_name=command_string*

Switches

/? Displays on-screen help including syntax, switches, etc.
/reinstall Installs a new copy of DOSKEY even if one is already installed.
/bufsiz=*nnnn* Specifies size of the buffer for command history and macros (default size is 1024 bytes).
/dmacs Displays list of all DOSKEY macros (output can be redirected, >, to file).
dhist Displays list of previous commands stored in buffer.
/insert New text on command line will be inserted into old text.
/overstrike New text will overwrite old text (default mode).

Errorlevel Not supported.

Network use Supported.

Using only about 3K of memory, DOSKEY is one of the handiest everyday features added to DOS in this release. Both of its key features were long needed and overdue.

Each of DOSKEY's major functions has its own set of keystroke commands. The up arrow recalls your previous command. Press it again and the command before that replaces it on the command line. You can scroll back, one command at a time, until you either reach the first DOS command issued during that work session or, if not that, the earliest one the buffer has room for. Because these are stored in RAM, the buffer is automatically cleared when you reboot your system. If you go too far back, the down arrow scrolls you forward. The PgUp and PgDn keys jump you to the top or bottom of the stack.

You can edit anything on the command line, whether it's a new command in which you notice a mistake or a recalled command you want to modify and run again. This is an even bigger time saver because, unlike the limited recall/edit capability of earlier versions by means of function keys backtracking on a line, this method doesn't wipe out the rest of the command, even temporarily. It's like

Table 20-2 DOSKEY keys and combinations used to move the cursor when editing commands. Cursor-left movements are nondestructive.

Configuration	Buffers	Bytes
128K-255K of RAM	5	2672
256K-511K of RAM	10	5328
512K-640K of RAM	15	7982

using your word processor—and as with the command history, the moves are pretty intuitive. See Table 20-2 for a list of the directional keys and what they do.

Well, you can't get away from those function keys entirely, it seems, and DOSKEY still supports the old way of doing things. But notice that they've slipped some new ones in there too, as shown in Table 20-3.

Creating macros involves a set of new commands that are hardly intuitive but fortunately few in number, and worth the effort to learn. A macro can be any DOS command or series of commands. They're like batch files even to the extent that you can use one of my favorite batch file tricks: having a macro execute a DOS command with the same name, but filling in all the switches and parameters without you having to remember and type them in. See Table 20-4 for some macro tips.

So why not just use batch files instead of macros? Once loaded—and you can load a bunch of them even in the default 1024-byte buffer—they run RAM fast. Try making macros for a couple of things you're doing now with batch files. You'll see.

Table 20-3 New DOSKEY function key command line actions, tailored to macro recall and use.

Key	Action
F1	Copies one character from previous command to the command line
F2	Copies previous command up to but not including the next letter typed
F3	Copies the remainder of the previous command
F4	Deletes the prior command up to your next keystroke
F5	Copies the current line to the buffer, but clears the command line
F6	Places an end-of-file mark (^Z) at the end of the current line
F7	Displays all command lines stored in the buffer
Alt-F7	Erases all command lines stored in the buffer
F8	Searches buffer for a command line that matches the one you've started. If match is found, DOSKEY automatically fills in remainder of the matching line
F9	Recalls previous command by number; first command is number 1, etc. Can't remember the number? Just press F7 for the list
F10	Displays all macros
Alt-F10	Clears all macros

Table 20-4 Special symbols necessary for many DOSKEY macro functions.

Entry	Action
$G (or) $g	Redirects output to file or device, equivalent to >
$L (or) $l	Redirects input from device or file, equivalent to <
$B (or) $b	Pipes output as input to other command, equivalent to |
$T (or) $t	Command separator, use either when creating macros or entering commands directly
$$	When dollar sign character is required, equivalent to $
$1-$9	Replaceable parameters, equivalent to batch file %1 through %9
$*	Replaceable parameter, roughly equivalent to $1 - $9, except *everything* typed on command line after macro name will be substituted for the $* character

And as an added benefit, a whole file of macros will probably use only one allocation unit (2048 bytes) on your hard disk. As separate batch files, each *one* would take the minimum 2028 bytes of disk space. Of nearly 60 batch files on my hard disk, most were under 48 bytes but a few went much over that, accounting for a waste of over 100K of hard disk space—which tells me I've got my work cut out for me one of these nights.

A simple macro to back up everything in the directory in which I kept the files for this book looks something like the line below:

```
DOSKEY BOOKBACK=XCOPY V:*.* /S/M
```

That line can either be entered from the command line for use of the remainder of the current session or included in a batch file, which, as I've indicated on the next line, could include as many macro entries as you'd care to make. You could load them all from a batch file that could be called from the AUTOEXEC.BAT file—or in a little DOSKEY batch if you don't want to run DOSKEY every session.

And look—I didn't even have to use any of those special characters to do my backup. You need them only when you want to get a little fancier with DOSKEY macros, like the one below that runs CHKDSK on whatever source disk\directory\etc., you specify. I specify $1 and then my target disk, $2, before running BACKUP.

```
DOSKEY BACKUP=CHKDSK $1$TCHKDSK $2$TBACKUP $1 $2 /A$3
```

And note that I've included the /a BACKUP switch so I don't forget and wipe out my previous backup instead of appending to it. But I still allow myself the opportunity to add more switches. Except for lines of over 256 characters (DOS's limi-

tation) and the ability to jump to labels, you can do virtually anything you can do with batch files, including using if/if not conditionals.

Also note that, in this last macro, I used the same name as one of the actual DOS commands I wanted to run, BACKUP, as my macro name. This is perfectly legal in DOSKEY macros. Even if the DOS program—in this case BACKUP—was in the path or even the default directory, DOSKEY looks *first* to its macros for a match and then and only then to DOS. Powerful stuff this DOSKEY macro business.

DOSSHELL

External (.COM)
Greatly enhanced!

Function Starts the DOS shell.

Usage Can be executed from the DOS command line or from batch files.

Origin Version 4.0.

Format DOSSHELL [/t[*res*[*n*]]][/b] (to start shell in text mode)
or DOSSHELL [/g[*res*[*n*]]][/b] (to start in graphics mode)

The variable *res* specifies the screen resolution category. Use l for low, m for medium, and h for high. The default value depends on hardware. The variable *n* specifies screen resolution where there is more than one choice within a category.

Switches

- /? Displays on-screen help including syntax, switches, etc.
- /t Starts shell in text mode.
- /g Starts shell in graphics mode.
- /b Starts shell using black-and-white color scheme.

Errorlevel Not supported.

Network use Not supported.

Not new in the sense of being a whole new function, DOSSHELL is nevertheless greatly improved, now adding task swapping to DOS's bag of tricks. First seen in DOS 4.0, the shell then was opened via a convoluted batch file that was created during installation by the SELECT (installation) program. In version 5x, it's all part of the 214K DOSSHELL.EXE file; a much more polished package any way you look at it.

It's also a big package, including the six files shown in Fig. 20-6, which together total over 400K, including DOSSWAP.EXE, the task switcher, which can be executed only by the shell.

```
DOSSHELL.COM      4,505 .a..
DOSSHELL.EXE    230,388 .a..
DOSSHELL.GRB      4,422 .a..
DOSSHELL.HLP    137,646 .a..
DOSSHELL.VID     15,942 .a..
DOSSWAP .EXE     18,548 .a..
```

20-6 A listing of the six primary files associated with the new DOS shell, requiring over 400K of disk space.

DOSSWAP

External (.EXE)
NEW!

Function The task switcher for DOSSHELL.

Usage Can be executed only by the shell.

DRIVER.SYS

External (device driver)

Function Enables the use of a logical drive designator to refer to an unsupported physical disk.

Usage Can be installed only from the CONFIG.SYS.

Origin Version 3.2.

Format DEVICE = [*d:*][*path*]DRIVER.SYS/d:*number* [*switch(es)*]

Switches

/d Specifies the physical drive number. Allowable values are 0−127 (0=A, 1=B, 2=C, etc.).

/c Indicated drive can sense if the drive door is open (change line support).

/f *factor* Specifies the drive type.

Values currently supported are as follows:

0 160K/180K/320K/360K
1 1.2 Mb
2 720K (any disk size)
7 1.44 Mb (3 1/2-inch only)

Note that if /f:*factor* is used, /h:, /s:, and /t: switches are not used. On the other hand, if /f:*factor* is not used, all three of those remaining three are normally required unless the default value specifically applies.

/h:*heads* Specifies the number of heads in drive (1−99). The default is 2.
/s:*sectors* Specifies the number of sectors per track (1−99). The default is 9.
/t:*tracks* Specifies the number of tracks per side (1−999). The default is 80.

Errorlevel Not supported.

Network use Not supported.

An intriguing driver, DRIVER.SYS can make one ordinary floppy drive act like two or more drives, or provide support for a variety of rather unconventional drives DOS or the ROM BIOS did not anticipate.

A problem many of us are faced with since the advent of the 3 1/2-inch floppy disk is a mix of drive sizes, typically one each of the 5 1/4-inch and 3 1/2-inch varieties. Which is fine until you want to copy something to the same size disk. DISK COPY is not the answer unless you want a mirror image of the entire contents, but you can't use COPY or XCOPY with the same drive for both the source and target disk unless you only have one drive. But you can use a single drive with DRIVER .SYS because it can make one physical drive answer to either of two (or more) logical drive designations. For instance, I added this line to my CONFIG.SYS

file:

```
DEVICEHIGH = C:\DOS\DRIVER.SYS /D:1/F:2
```

which gave my drive B: (/1) an alias—the next available letter of the alphabet, which in this case was G. Then I used XCOPY to copy files from drive B: to drive G:—the same physical drive, but treated as a separate device. With my source disk in B:, I entered xcopy b:*.* g: at C>, and this is how it went from there:

```
Insert diskette for drive G: and press any key when ready
Insert diskette for drive B: and press any key when ready
Reading source file(s)...
Insert diskette for drive G: and press any key when ready
```

Then I typed ONCEOVER.SE2 and ONCEOVER.SE1 at the B>, and got:

```
Insert diskette for drive B: and press any key when ready
2 File(s) copied
Insert diskette for drive G: and press any key when ready
```

I could have used COPY, but it's much slower and requires a lot more disk swapping. Assuming you have enough memory to work with, XCOPY sucks everything up at once, which generally gives it a fair advantage over COPY. But even with XCOPY, it took over two minutes to copy these two files, which total about 150K. That seemed slow so I tried copying the same two files to a RAM drive, then back to a fresh disk in the same drive without all the hocus pocus, and knocked 37 seconds off the time. But it's interesting to note that DRIVER.SYS can be used only with floppy drives and not with hard disks.

DRIVER.SYS also allows you to address and use nonstandard drives as ordinary DOS drives. Here too, however, DRIVER.SYS might not be the best answer. In my experience, most of the unfamiliar formats that aren't supported by DOS come with their own proprietary drivers. When 3 1/2-inch floppies first became available for upgrading older machines, DOS didn't support the 720K or 1.44K formats, which were generally shipped with their own proprietary device drivers. DOS introduced the DRIVPARM option in version 3.2 for dealing with these on older machines where they weren't supported by the ROM BIOS. When such drivers are supplied, DRIVER.SYS isn't required.

Most drives like these also require their own proprietary interface cards, whether or not special drivers are supplied. DRIVER.SYS can't take the place of such special hardware support nor can it be used to add 1.44 Mb capability, for instance, to a system where the floppy disk controller is limited by design to some lesser format capacity.

Few users will probably ever need to use DRIVER.SYS, but it's nice to know it's there. Particularly given the seemingly endless list of new storage media formats you see these days.

DRIVPARM

Internal to IO.SYS

Function Defines parameters for nonstandard block devices when starting DOS.

Usage Can be used only in the CONFIG.SYS.

Origin Version 3.2.

Format DRIVPARM = /d:*n* [/c] /f:*factor* [*switch(es)*]

Switches

/d:*n* Specifies a physical drive number from 0-255, 0=A, 1=B, etc.

/c Specifies if a drive can sense whether the drive door is closed.

/f:*factor* Specifies drive type.

Supported drives are:

0 160K/180K/320K/360K
1 1.2 Mb
2 720K 3 1/2-inch disk (default)
5 Hard disk drive
6 Tape drive
7 1.44 Mb 3 1/2-inch disk

/h:*heads* Specifies the maximum head number. Valid values from 1−99. The default value varies depending on the /f:*factor* specified.

/i Specifies 3 1/2-inch floppy drive type not supported by your system.

/n Specifies a nonremovable block device.

/s:*sectors* Specifies the number of sectors per track. Valid values are from 1−99. The default value varies, depending on the /f:*factor* specified.

/t:*tracks* Specifies the number of tracks per side. The default value varies, depending on the /f:*factor* specified.

Errorlevel Not supported.

Network use Not supported.

Note that this function isn't available in PC-DOS. DRIVPARM comes into play mainly when you add drives to your machine that neither DOS nor the machine's designers anticipated. Typically, to add a 3 1/2-inch floppy to a machine that predates their popularity you must include DRIVPARM to your CONFIG.SYS. With many machines, while DOS will read 720K 3 1/2-inch disks, you can't format them except as a 360K unless you use DRIVPARM with the appropriate parameters. It's usually easy, however. Here's one I use:

```
DRIVPARM = /D:0
```

Here, it's enough just to tell DRIVPARM which drive(s) need help because 720K 3½-inch is the default.

Two of my machines use the above CONFIG.SYS line—and one of them isn't even an older machine, but rather a well-known laptop that came factory equipped with 3½-inch drives and doesn't have a ROM BIOS that properly supports them. Which is something you don't always know when you buy a machine, because the OEM DOS can be gimmicked to do it instead. But when you upgrade to a different DOS release, you suddenly have these 360K 3½-inch drives until you install DRIVPARM.

When you start adding things like tape drives, it gets little more complicated. There you usually have to spell out the parameters in detail. This data should all be in the spec sheets that come as part of the documentation for a drive. A tape with 10 tracks of 99 sectors configured as drive E:, for example, would typically require the following CONFIG.SYS line:

```
DRIVPARM = /D:4 /F:6 /H:1 /S:99 /T:10
```

While an /i switch is provided for use with machines—including some IBM machines—that have ROM BIOS routines that don't support 3½-inch disks, I have yet to find one that actually requires it. Just remember it's there if all else fails.

ECHO

Internal

Function Turns the command echo on or off—used typically with batch files. Also used to display messages.

Usage Of use primarily in batch files.

Origin Version 1.x.

Format ECHO ON
or ECHO OFF
or ECHO (entered without parameters, it displays current ECHO setting)
or ECHO [*message*]

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

ECHO is a versatile tool that's often overlooked—or at least under-used. Most users know that ECHO OFF can be used at the top of a batch file to keep all the various commands and nonessential DOS messages from cluttering the screen. Beginning with version 4.x, you can even make the ECHO OFF line—or any other line—invisible by simply putting an @ in front of it.

You can also ECHO nothing (just a blank line) by putting a period *immediately* following ECHO. To slip some totally blank lines into a batch file that doesn't have the echo turned off—to separate two messages, for instance—you could put in something like:

```
@ECHO.
@ECHO.
@ECHO.
@ECHO.
```

I use ECHO a lot in conjunction with ERRORLEVEL. Here is an excerpt from one batch file that tests the error level and uses ECHO to display an appropriate message:

```
IF ERRORLEVEL 0 ECHO Backup of files in %1 completed satisfactorily
IF ERRORLEVEL 1 ECHO No files found matching backup criteria in %1
IF NOT ERRORLEVEL 0 GOTO ERROR
```

You can ECHO almost anything you want displayed—or don't want displayed if

you'll excuse the seeming contradiction. There's one notable exception, however. You can't display piped or redirected characters or strings.

As for echoing things you don't want displayed, one of the handiest but little known tricks you can do with ECHO is use it to send control strings—those pesky escape (ESC) sequences—to your printer. It isn't always convenient to let your word processor do it and you really can't do it easily from the DOS prompt.

What you'd like to send through to your printer would be something like ECHO (Esc)*string* >PRN, but that Escape character is pretty slippery. As soon as you touch the Escape key, the command line clears, cancelling the command the way it's supposed to. If you try to get sneaky and hold down the Alt key while you type 27 (the decimal equivalent) on the number pad, DOS isn't fooled and shoots you right back to the prompt.

You can do it, however, from a batch file using ECHO and little redirection, as I've done here with a little one-liner I call ESC.BAT:

```
ECHO %1 >PRN
```

My favorite word processor—XyWrite—takes things like Escape key entries right in stride. Unfortunately, many don't—but DOS's slick new EDIT does. If yours does, fine. If not, there's another way to go about it, by using DEBUG, and you'll find the instructions for creating this program described step-by-step in the DEBUG chapter earlier in this book. Also, those using ANSI.SYS might find ECHO a handy means of passing escape sequences to the driver from batch files.

I use ECHO in another way, as well—to read the date from the system calendar into a file. A batch file in this case, which, as described in the chapter on batch files and DOSKEY macros, I then use to perform a number of housekeeping chores on my system.

```
ECHO|DOS\MORE|DATE >DOS\BATCH\LASTDATE.BAT
```

The piping (|) and MORE command here simply add the necessary carriage return to close the DATE command. So there are many uses for ECHO, both for things you want to see and things you don't.

EDIT

External (.COM)
NEW!

Function Starts the new MS-DOS editor for creating or changing ASCII text files.

Usage Can be executed from the DOS command line or from batch files.

Origin Version 5.0.

Format EDIT [*d:*][*path*][*filename*]

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

After years of kicking EDLIN around, Microsoft has finally come up with a pretty decent editor. It has pull-down menus and an automatic help line that explains the currently highlighted choice. It also has dialog boxes and a lot of built-in features—like not letting you exit and leave an unsaved file open without pausing and prompting you, and dumping your file only if you really want it to—a far cry from old EDLIN.

EDIT doesn't have everything I'd like to have seen in it, however. In this release, at least, it lacks any formatting capability—a seemingly relatively simple thing considering the effort that's gone into it. It comes that close to really being a dandy.

Still, even as it stands, EDIT is really handy for creating and editing batch files, and I got a lot of use out of it while I was working on this book, and not just because I felt I had to use it. It's just very handy. And so easy. I used it for weeks before I even looked at the documentation—and then only to be sure I wasn't overlooking something in the chapter in this book where I examine it in detail. Easy and intuitive—eat your heart out, EDLIN.

EDLIN

External (.EXE)

Function Text editor for creating or modifying files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0 and later.

Format EDLIN [*d:*][*path*][*filename*][*.ext*][*switch*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/B Causes end-of-file markers within a file to be ignored. Version 2.0 and later.

Errorlevel Not supported.

Network use Not supported.

Dating to a time before there was real application software for the infant PC—before word processors, before most any kind of 3rd-party software aftermarket—EDLIN had a rating as near a zero as any of the DOS utilities. It has sometimes been described as the text equivalent of DEBUG, but mostly, I suspect, by people who don't understand DEBUG. There's no comparison.

EDLIN was clumsy then and it's clumsy now—and made to look all the clumsier along side MS-DOS's new EDIT. If not for Microsoft's penchant for continuing to support the archaic, the arcane, and even the unsupportable, I can see little reason for it even surviving into this release.

Given that, I'm not even going to dignify EDLIN by giving space to its clumsy syntax and other even less endearing features. If you just have to know, Microsoft still devotes over 15 pages to it in their DOS 5.0 documentation.

If you haven't used EDLIN for a while—or if you've never used it—sometime when you've got nothing better to do you might want to dig it up and play with it a little. If nothing else, it will make you appreciate how far, how very far, DOS has come.

EGA.SYS

External (device driver)

NEW!

Function Saves and restores the display if you use task switching and have an EGA monitor.

Usage Can be loaded only from the CONFIG.SYS.

Origin Windows 3.0

Format DEVICE = [*d:*][*path*]EGA.SYS

Switches None.

Errorlevel Not supported.

Network use Not supported.

According to the documentation, you *must* use EGA.SYS if you have an EGA display *and* intend to use DOSSHELL's new task switching function. Failure to do so will cause the system to hang and it doesn't matter whether you're using the shell in graphics or text mode. It will still hang.

There are exceptions, but probably not many. If you're using Windows 3.0 with an EGA, you'll probably find that Windows has already added EGA.SYS to your CONFIG.SYS. Also, I did find that one device driver I was using on one machine served the same function and, in that instance, the new DOS EGA driver was not needed. If you should observe an error message during bootup when the system tries to load the driver, you might want to try removing it. If it hangs when you use the task swapper, put it back.

There's also one interesting note regarding the use of EGA.SYS if you're also loading a mouse driver from the CONFIG.SYS. It generally takes less memory to load EGA.SYS before you load the mouse driver than after it.

EMM386

External (.EXE)

NEW!

Function Enables or disables EMM386 expanded memory emulation (286 or higher systems only). Also enables or disables Weitek Coprocessor support.

Usage Must be installed from CONFIG.SYS but can be reactivated, etc., from the DOS command line or in batch files.

Origin Version 5.0.

Format DEVICE = EMM386.EXE *options*
or EMM386 *switch(s)*
or EMM386 (when entered without parameters, displays the current EMM386 status)

Switches

/? Displays on-screen help including syntax, switches, etc.
on Activates EMM386 device driver.
off Disables EMM386 device driver.
auto Puts EMM386 device driver in "auto" mode.
w = on Enables Weitek Coprocessor support.
w = off Disables Weitek Coprocessor support (default).

Errorlevel Not supported.

Network use Not supported.

Playing a key role in DOS 5.0's new support for memory beyond 640K, not only for its own use but for use by other software as well, EMM386 emulates expanded memory. Whether you start with expanded memory or emulate it is more a game played by software developers than something that has any special meaning. The way DOS plays the game is to start with extended memory because that's what Windows 3.0 uses and also where DOS has to "find" that extra 64K HMA segment it can access beyond its traditional 1-megabyte limit with 286 and higher machines. HIMEM.SYS sets the stage for that by turning excess memory—anything beyond 640K—into extended memory.

Many programs require expanded memory. These can't use extended memory, so what EMM386 does is take some of that extended memory and use it to *emulate* expanded memory so those programs can use it. Programs don't know or care where the memory comes from as long as it's there and everybody's happy.

To use EMM386, you must install it from the CONFIG.SYS—whether or not you have immediate need for it. Once installed, it can be activated or reactivated

from the command line. With the exception of the auto mode, the switches are fairly self explanatory. The command EMM386 AUTO enables Weitek Coprocessor support automatically when and as needed.

EMM386 can, if not currently active, be activated or reactivated (if installed from the CONFIG.SYS) from the command line by entering the command EMM386 ON. It can't be deactivated once expanded memory is actually in use, however.

EXE2BIN

External (.EXE)

Function Converts .EXE files to binary (.COM) file format.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.1 and later.

Format EXE2BIN [*d:*][*path*]*exe_filename*[*.ext*] [*bin_filename*]

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Supported.

By Microsoft's admission, EXE2BIN is included with MS-DOS mainly as a courtesy to software developers. This is one you won't need or even want to play with unless you're into programming. In fact, although furnished with earlier DOS releases, EXE2BIN was available for DOS 4.x only on the Utilities disk supplied with the Disk Operating System Technical Reference manual.

For what it's intended to do, EXE2BIN does the job, but because it isn't meant for everyday DOS users but rather for a very specialized group, I won't discuss its usage in greater detail here. And because applications that require EXE2BIN most generally require support by other utilities not even furnished with MS-DOS, I don't keep this one in my DOS directory. You might not even want to keep it on your hard disk.

EXIT

Internal to COMMAND.COM

Function Exits a secondary command processor, returning control to the primary command processor (COMMAND.COM).

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format EXIT

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Supported.

You've no doubt used this one many times, possibly without even realizing that you were actually killing a copy of COMMAND.COM—or really caring, for that matter. Still, it's a dirty job that must be done so you can terminate a program that's loaded a new copy of COMMAND.COM, so Microsoft created EXIT.

If by any chance that secondary processor has been loaded with the /p switch (for permanent), you're stuck until you reboot. Programs generally don't, however, and as long you don't you should have no problem.

EXPAND

External (.EXE)

NEW! and undocumented

Function Expands MS-DOS files from distribution disks.

Usage Used by SETUP program when installing DOS 5.0 or upgrading from distribution disks, it can also be executed from the DOS command line or in batch files.

Origin Version 5.0.

Format EXPAND [*d:*][*path*]*source_file*[.ex_]] [*d:*][*path*][*target_file.ext*]

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

Another new addition, brought into being by Microsoft's move to file compression on the 5.0 distribution disks, EXPAND is intended mainly for decompressing the DOS files when installing them on your hard disk (or floppies), as part of the standard installation process. It's also used when decompressing the DOS 5.0 network redirectors that aren't part of the basic MS-DOS OEM original installation or more generic MS-DOS upgrade packages, but rather marketed separately.

EXPAND isn't a generic file decompression utility and shouldn't be used to expand any compressed files other than those on the MS-DOS distribution disks. There are a number of different compression/decompression schemes, each based on the use of some particular proprietary set of algorithms, which, by their nature, are not compatible. Still, I think it's a good command to keep handy, particularly if, after erasing a bunch of unused DOS files to free space on your hard disk, you find later that you do need something you zapped after all. By using EXPAND.EXE, you can expand the compressed files from the distribution disk selectively, rather than going through the whole SETUP process again.

It's easy, by the way, to tell which files are in compressed form on the distribution disks and which ones aren't. The compressed files have the last character of the extension replaced by an underline character, as in .CO_ and .EX_. You *must* use the complete proper extension for the target file if included at all when using EXPAND. Failure to do so will result in an error message even if the file that's valid except for the extension is on the directory. If you expanded properly, therefore, and you type:

```
CHKDSK
```

at the C>, you might receive the following DOS message:

```
Unknown command "CHKDSK"
```

The destination might be only a filename if the *source_file* is a single filename. To expand multiple files to a different directory and keep the original filenames, specify only a directory as a destination. Other than that, the use of this command is pretty straightforward.

EXPAND is capable of creating bootable disks, completely bypassing the normal SETUP routine. I wouldn't recommend this, however, even for creating bootable floppies. You're much better off using either the SYS command or the /s switch when formatting (FORMAT [*d:*] [*other_switch(s)*]/s). I mention it here mainly for informational purposes.

There are two real drawbacks to using EXPAND for the creation of bootable disks. The first is that it's a three-stage process that must be accomplished in the following order:

```
EXPAND [d:]IO.SY_ [d:]IO.SYS
EXPAND [d:]MDDOS.SY_ [d:]MSDOS.SYS
EXPAND [d:]COMMAND.CO_ [d:]COMMAND.COM
```

Keep in mind that, as when creating any bootable disk, no other files can have been copied to it since it was formatted or the disk must have been formatted using the /b switch. The second drawback is that, expanded in this manner rather than through any of the recommended procedures, the r, s, and h attribute bytes of the two system files are not set, as shown in Fig. 20-7.

This latter objection can, of course, be overcome by using the ATTRIB command to set these attributes, but it's still a cumbersome procedure at best.

```
COMMAND .COM    46,246 .a.. 12-13-90   4:09:18 am
IO      .SYS    33,044 .a.. 12-13-90   4:09:18 am
MSDOS   .SYS    37,506 .a.. 12-13-90   4:09:18 am
```

20-7 A listing showing a complete DOS system created by means of the EXPAND command, rather than INSTALL (not recommended).

FASTOPEN

External (.EXE)

Function Provides quick access to recently used subdirectories and files.

Usage Can be installed from the CONFIG.SYS or loaded from the DOS command line or batch files.

Origin Version 3.3 and later.

Format FASTOPEN *d:*[=([*nnn*][,*mmm*])][*switch*]
or INSTALL=[*d:*][*path*]FASTOPEN *d:*[=([*nnn*][,*mmm*])][*switch*]
with the following arguments: *nnn* stands for the number of files for FASTOPEN to work with (range 10−999), and *mmm* is the number of file extent entries (range 1−999). Version 4.0 and later only.

Switches

/? Displays on-screen help including syntax, switches, etc.

/X Loads program into expanded memory. With version 4.x of PC-DOS, this switch worked only when using a genuine IBM expanded memory board and IBM device driver XMA2EMS.SYS.

Errorlevel Not supported.

Network use Not supported.

There are, if cat lovers will forgive me, many ways to skin a cat. In other words, there are often many ways to accomplish much the same thing in DOS. FASTOPEN is one, BUFFERS another. Then there's SMARTDRV.SYS, which doesn't even try to do the same job as either of those but gives you pretty much all the benefits of both. And then some.

FASTOPEN caches filenames and that's all it does. Just filename—the same way buffers squirrel away disk data in little sector-size (512-byte) mini caches. A good disk cache does it all.

Granted, SMARTDRV.SYS or any serious, heavy-duty disk cache requires a lot more memory. FASTOPEN, aside from the roughly 4K it takes to load, requires only 48 bytes for each file you specify (*nnn*) when you load the program, or about 20 filenames for each additional kilobyte of memory you give it. And unlike a full disk cache, giving FASTOPEN more memory won't make it run one bit faster.

I use disk caching on all of my machines and, frankly, if FASTOPEN makes them run any faster, I've yet to see it. Without disk caching, I'm sure it would be a different story. If your memory resources are too limited to support full disk caching, then you might see a noticeable improvement when using FASTOPEN. Don't just take the manual's word for it on this one, though.

FC

External (.EXE)

Function Compares two files or sets of files, displaying any differences between them.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later. Not available in all implementations of DOS.

Format FC [*d:*][*path*][*filename1*] [*d:*][*path*][*filename2*] [*switch(es)*]

Switches Note that these switches precede filename arguments.

/? Displays on-screen help including syntax, switches, etc.
/a Abbreviates output of ASCII comparison.
/b Forces binary comparisons with no resync attempt after mismatch.
/c Ignores case of letters.
/L Compares files in ASCII mode (default except with .EXE, .COM, etc.
/Lb*n* Sets internal line buffer to *n* lines, and aborts if more lines different.
/n Displays line numbers in ASCII comparisons.
/t Does not expand tabs to spaces.
/w Causes FC to compress tabs and spaces.
/*nnn* Specifies number of lines that must match after mismatch.

Errorlevel Not supported.

Network use Supported.

Even at first glance, this is a far more powerful utility than COMP, another of those throwbacks to another age. FC adds a number of whistles and bells. It also requires a lot more memory. In addition to the space required to just load the program—2/3 bigger than COMP—it requires enough additional RAM to hold 100 lines.

As is the case with so many other DOS commands, you shouldn't let all the switches and options intimidate you. I don't use this one often enough myself to remember them all, so when I do need to compare a couple of files I just run it without any switches. FC is smart enough to give a pretty useful display of either ASCII or binary data—files with .COM, .EXE, .SYS, .OBJ, .LIB, and .BIN extensions default automatically to binary mode. When comparing text files in its default state, FC displays enough of the differing text passages to show not only just the differences, but the context of the statements. Binary, of course, is just binary any way you slice it.

If the defaults don't tell me what I need to know, *then* I look up the switches. I usually don't have to, though. Just plain old unembellished FC most often comes through just fine for what I need most of the time.

If you do use FC often and you do need some of its switches, and pretty much the same ones every time, FC would be a real good macro candidate for DOSKEY. Here, I've created one that makes the comparison insensitive to case and abbreviates the display:

```
DOSKEY FC=FC $1 $2 /a/c.
```

FCBS

Internal to IO.SYS

Function Specifies the number of file control blocks that can be used when file sharing is implemented.

Usage Can be used only in the CONFIG.SYS.

Origin Version 3.x.

Format FCBS=*nnn* (1–255, the default is 4)

Switches None.

Errorlevel Not supported.

Network use Not supported.

Installable only from the CONFIG.SYS, FCBS is another one that most users will never have to deal with—so few that the DOS documentation gives little information beyond "You should use the FCBS command only if an application requires you to do so . . ." It goes on to explain that few do and that it's mainly for use with some older applications and networking environments. I've never needed it and chances are you won't either, unless you're into networking.

FDISK

External (.EXE)

Function Configures a hard disk for MS-DOS use.

Usage Should be executed from the DOS command line.

Origin Version 2.x.

Format FDISK

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Not supported.

This is one I think should be furnished on a separate utilities disk—preferably a bootable floppy—rather than installed automatically on your hard disk. The likelihood you'll ever have to use it is pretty much dependent on whether you either buy an additional or replacement hard disk that isn't dealer installed or try to recover a hard disk that's had a totally, nonrecoverable catastrophic failure.

This is not to recommend against either, and I've used both quite successfully. Much of this book, in fact, was written on a machine with a hard disk that's been resurrected more than once, from the low-level format right on up. But I keep FDISK on a bootable floppy because usually when I need it the machine I need it for doesn't *have* a hard disk—or doesn't think it has at that point.

Don't get the idea that FDISK is one of those commands to be afraid of, either. Just because you start the program doesn't mean it's going to wipe out your hard disk if it's healthy. FDISK doesn't do *anything* without checking to see exactly what it is you want done, as shown on the first screen you'll see, in Fig. 20-8.

From the opening screen, you can poke around a little deeper if you like. Option one, the default, or any of the first three options could get you into trouble, but even they won't hurt you without prompting you to make sure that you really want to do it. Option 4 is perfectly safe and shows something like what's shown in Fig. 20-9, which isn't so scary after all. Not when you know it isn't going to jump right out and eat your hard disk—not without fair warning, anyway.

```
                    MS-DOS Version 5.00
                Fixed Disk Setup Program
          (C)Copyright Microsoft Corp. 1983 - 1990

                        FDISK Options

Current fixed disk drive: 1
Choose one of the following:
1. Create DOS Partition or Logical DOS Drive
2. Set active partition
3. Delete DOS Partition or Logical DOS Drive
4. Display partition information

Enter choice: [1]

Press Esc to exit FDISK
```

20-8 The first screen displayed by FDISK. No action taken at this level is irreversible, and option 4 can provide useful information.

```
                    Display Partition Information

Current fixed disk drive: 1
Partition Status   Type    Size in Mbytes   Percentage of Disk Used
 C: 1        A    PRI DOS        32              78%
    2             EXT DOS         9              22%
Total disk space is   41 Mbytes (1 Mbyte = 1048576 bytes)

The Extended DOS Partition contains Logical DOS Drives.  Do you
want to display the logical drive information (Y/N).....?[Y]

Press Esc to return to FDISK Options
```

20-9 The second level FDISK screen, after choosing 4 from the first screen, shown in Fig. 20-8.

FILES

Internal to IO.SYS

Function Sets the number of open files MS-DOS calls can access at one time.

Usage Can be used only in the CONFIG.SYS,

Origin Version 2.x.

Format FILE = *nnn* (valid numbers are from 8, the default, to 255)

Switches None.

Errorlevel Not supported.

Network use Not supported.

Installable only from the CONFIG.SYS, this is one of DOS's more innocuous but poorly understood commands. You hear a lot of different numbers kicked around for this one, but the bottom line is that it really isn't critical as long as you specify a high enough number (if you don't, just go with the eight you get by default).

Unlike BUFFERS, specifying too many won't slow your system down. But increasing the number does take a little more memory—an extra 39 bytes or so for every additional one you specify, so you can't just go with "more is better." Some 3rd-party search utilities recommend that you provide for 30 or 40 files, but I've never found it necessary to go that far.

Working in a multitasking environment—Windows or DESQview for instance—you'll probably need more than just the eight. But even there I wouldn't go much more than 20. If you run short, some day you'll know it when DOS just won't open another file until you close the one you've got open. At that point, either DOS or your application will probably give you some sort of error message in case you miss the point. In any case, don't sit there worrying or rush to change the number now.

FIND

External (.EXE)

Function Searches for specified stings in text file(s).

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format FIND [*switch*]"*string*" [*d:*][*path*][*filename.ext*]...

Note: wildcards *are not* allowed with FIND. Additional filenames (separated by a single space) can be specified on the command line, however.

Switches (can precede or follow arguments)

- /? Displays on-screen help including syntax, switches, etc.
- /c Displays only the number of lines that contain a match in each file.
- /i Specifies that the search won't be case-sensitive.
- /n Precedes each line with its relative line number in the file.
- /v Displays all lines *not* containing the specified string.

Errorlevel Not supported.

Network use Supported.

While a frequently needed function, DOS's FIND filter utility is still severely limited by its documented inability to handle wildcard entries either for filenames or extensions. To me, that rather defeats the purpose—if I knew what file I was looking for, I'd just open it and let my word processor find the text string.

For that matter, even my word processor can search a whole directory if I want it to. But not FIND. And FIND is fussy. You have to enclose the string you want it to look for in quotes. Just making it the first parameter after the command doesn't cut it. The new /i switch helps by at least allowing you to turn off its default case sensitivity.

To make FIND really work the way it should, however, it needs help from one of DOS's rather more obtuse commands—FOR. Not many users really use the FOR command, it seems. And many of the uses you see documented seem a little pointless, but here's one that isn't—a little one-line gem I call FIND.BAT:

```
FOR %%A IN (%2) DO FIND %1 %%A %3
```

For an explanation of how it works, you'll have to look ahead a page or two to where I talk about the FOR command. But remember that FIND can't handle wildcards. Look at this, though:

```
FIND MS-DOS FIVE\*.*
```

I just told FIND (through FIND.BAT) to locate every mention of MS-DOS in

every file in a directory—and using the FOR command, it does it. Again, I've used one of my pet tricks, giving the batch file the same name as the DOS command I want it to run, and arranging it so the command line parameters or arguments are entered in the same sequence as running the command directly. I took the liberty of sticking the quote marks in the batch file so I don't have to type them on the command line every time. But that's optional, and, to keep the syntax the same as if using the DOS command directly, you might do well to leave them out.

But the important difference here is that FIND.BAT actually does what just plain FIND can't do. And here's the proof—for the following command:

```
FIND 'MS-DOS FIVE:EXTENDED
```

here is part of just one of the many finds it made in that pass, complete with the actual filename it found the following mentions in:

```
[39]    655360 bytes available to MS-DOS
[50]    655360 bytes available to MS-DOS
[56]    MS-DOS resident in High Memory Area
[121]   The MS-DOS Editor recognizes keystroke combinations familiar to
        users of other Microsoft programs (such as Microsoft Word) as well as
        WordStar key combinations. WordStar keystrokes that can be used in
        the MS-DOS Editor are listed in the right column of the three-column
        keystroke tables.
```

Not only the specific file, but line number the matches were found in, and in each case enough of the surrounding text to put the reference in its proper context. All of a sudden FIND gets interesting.

Unfortunately, the more limited set of variables available for use in DOSKEY macros doesn't seem to support this bit of razzle-dazzle quite as well, but it can be done. While DOSKEY supports a special code for redirection, I found that I had to insert the regular redirection arrow into the command manually to keep DOSKEY from balking when I didn't want the output redirected. Double redirection arrows are required to append output from multiple files.

So don't just write FIND off as being as clunky as it looks at first glance. You can do a lot with it. It's just too bad that, as much as Microsoft has done that's good in version 5.x, they didn't do a little more to help.

FOR

Internal to COMMAND.COM

Function Executes a specified command for each file in a set of files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.x.

Format FOR %%c IN (*set*) DO *command* [*arguments*] (in batch file)
or FOR %c IN (*set*) DO *command* [*arguments*] (from command line)

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

For every power user who swears by this one, there are probably a hundred, possibly a thousand, other users who avoid it like the plague. It doesn't happen to be one of my favorites either, but there are times when it can really come in handy.

The thing that probably puts most users off is its seemingly convoluted syntax: those percent signs—sometimes one, sometimes two—and mandatory parentheses. So let's start by trying to decipher it. Take the command line version, although both work the same. Read the batch file I showed you for FIND, and it says:

```
FOR %c IN (%2) DO FIND "%1" %c
```

In other words, do the following action to the next file in this group, then loop to the next file, etc., until all specified files have been done. Or, in the order of the command itself:

IN the next file in this group, look for "*string*" in that file

You could carry it one step farther, adding another variable (%3) at the end to allow the output to be redirected—I did in mine and it's often useful. In the batch file, I also added some switches for FIND, but they don't affect the way FOR works, so for the sake of simplicity I didn't show them here.

Notice, however, that I have the second parameter (%2) specified on the line before the first one (%1). This is no accident. The second parameter you have to give FIND is where to look, which, in this case, is information I have to pass along to FOR rather than giving it to FIND. FOR does that when it passes %c.

The %c variable can use any character. In DOS 5x, it can be any letter of the alphabet *or* a number from 0 through 9—earlier versions could use letters only. In

this usage, a letter works best and any old letter will do—as long as you have only one variable for DOS to sort out.

Here, as you see, I used only the single percent sign. To use this line in a batch file, all I would have to do is use double percent signs instead (%%). DOS strips the second one when it runs the batch file, but if it isn't in the batch file DOS can't read it as a variable.

FOR has slipped from favor even among many old-time power users in recent years. It came on the scene when DOS was still pretty crude—before the days of XCOPY and REPLACE and some of today's more powerful commands that can cut their way through subdirectories with a single slash. But as you can see it can still be very handy, breathing new life into little urchins like FIND.

FORMAT

External (.COM)
Major changes!

Function Initializes floppies and hard disk for use by DOS.

Usage Generally used only from the DOS command line.

Origin Version 1.0 and later.

Format FORMAT *d:* [*switch(es)*]

Switches

/?	Displays on-screen help including syntax, switches, etc.
/s	Transfers DOS System files during format process.
/b	Reserves space for System but does not transfer.
/v:*l*	Adds volume label *l*.
/q	Quick format. Nondestructive; erases FAT and root directory only.
/f:*n*	Specifies formatted capacity if other than drive default.
/u	Specifies unconditional format which destroys any and all data existing on the disk. No data can possibly be recovered if used.

The following switches are also still supported by DOS 5.0, but their use has largely been supplanted by the new, more comprehensive /f: switch:

/1	Formats disk as single sided.
/4	Formats 360K disk in 1.2 meg drive.
/8	Formats 8 sectors per track (default is 9 or 15).
/n:*n*	Specifies number of sectors to format per track.
/t:*n*	Specifies number of tracks to format.

Errorlevel

0	FORMAT successfully completed.
3	FORMAT terminate by user (Esc or Ctrl−Break).
4	FORMAT terminated due to error.
5	FORMAT terminated by user response of N to warning about formatting a fixed disk.

Network use Not supported.

One of DOS's most critical—and most maligned—utilities, FORMAT underwent a major overhaul in version 4.x and again in 5.0. Most of the changes in 4.x, though good, were cosmetic more than substantive. But in 5.0, the changes are not only pretty but highly significant. This new version of one of DOS's most despised but yet essential utilities can, for the first time, be run nondestructively on floppies, as well as a hard disk. In fact, the nondestructive mode is actually the normal default mode.

Indeed, FORMAT has come a long way since the days when it would reformat your hard disk at the slightest provocation. If you neglect to tell it which drive you want formatted, it aborts, telling you *Required parameter missing*. And even if you insist you want your hard disk formatted, it will pause to prompt you with:

```
WARNING, ALL DATA ON NON-REMOVABLE DISK
DRIVE C: WILL BE LOST
Proceed with Format (Y/N)?
```

It really is a kinder, gentler world. Those familiar with the Norton Utilities or any of several other 3rd-party format utilities will notice that DOS's new FORMAT operates a little differently (based on technology licensed from Central Point Software). FORMAT, while nondestructive in its default mode, does zero-out the root directory and File Allocation Tables (FATs). The data they contain, however, is copied to unused or erased filespace in a special way that allows the new UNFORMAT utility to find that data and copy it back to the FAT and root directory if need be, restoring the disk to its pre-reformatted condition.

This differs from the Norton Utilities and similar nondestructive formatters in that they don't zero-out the root directory and File Allocation Tables, rather treating any files in the root directory as if they had simply been erased (by discarding the first character of each filename). They do, however, keep their FAT entries intact.

It's especially important for anyone familiar with these utilities to understand this difference because while both systems have merit, the DOS UNFORMAT command won't—can't—unformat a disk reformatted with the Norton (or other similar) formatter *or vice versa*. The new DOS UNDELETE utility can at least partially restore a disk reformatted with the Norton formatter, but I wouldn't recommend mixing the two.

Without getting into the pros and cons of the new DOS FORMAT scheme, the fact that no data is retained in the FAT and root directory areas, but rather removed and stored elsewhere on the disk can sometimes create problems.

If, before reformatting, the disk has less free space in the file area than will be required to save the system data, then the disk won't be recoverable. Not by DOS or any other means (except laboriously on a sector-by-sector basis). You'll get fair warning, however—more than that, the new DOS will give you a chance to reconsider before proceeding, as shown here (where I tried to reformat a disk that was nearly full):

```
C:\FORMAT B:/f:720
Insert new diskette for drive B:
and press ENTER when ready...
Checking existing disk format
Saving UNFORMAT information
```

```
Drive B error. Insufficient space for MIRROR image file.
There was an error creating the format recovery file
This disk cannot be unformatted
Proceed with Format (Y/N)?
```

What to do? If keeping the UNFORMAT option open is important, you can check the directories, erase any files you're sure you won't ever need to recover, and try again. It doesn't take a lot of space because you're dealing with just the FAT and root directory data. The actual amount depends on the disk size and formatted capacity, but in the case of a 720K floppy as in the above example, about 7K of free space seems adequate (the size of the MIRROR.FIL if you've run MIRROR). Below is the screen when I ran FORMAT again after freeing 7K:

```
Checking existing disk format
Saving UNFORMAT information
Verifying 720K
85 percent of disk formatted
```

It should be pointed out, however, that unformatting a disk is not dependent on running the MIRROR utility. MIRROR should be run routinely—on a daily basis, preferably—on your hard disk. However, its use with floppies, while enhancing the overall chances of a full recovery after an accidental reformatting, is not essential. I personally recovered a number of reformatted disks without its help during the writing of this book.

Another significant change, seen first in 4.x but carried over into this release, is the *form factor* switch, which greatly simplifies and makes obsolete (but does not replace) several of FORMAT's more cumbersome and confusing options. As new disk sizes and capacities came into common usage, FORMAT added several poorly thought-out switches to deal with them: /1, /4, and /8. Then it jumped to specifying tracks and sectors—numbers few users could remember.

The form factor (/f:*nnn*) clears away the muddle by simply requiring that you specify the formatted capacity in kilobytes, which at least is something most users can remember. Valid format capacities include those shown in Table 20-5.

Table 20-5 Valid formats for 3½- and 5¼-inch disks.

For 5 1/4-inch disks	For 3 1/2-inch disks
160K, 180K	720K
320K, 360K	1440K (1.44 Mb)
1200K (1.2 Mb)	2880K (2.88 Mb)

Unfortunately, as a practical matter there is little that can be done to automate or simplify the job of formatting. You can write little DOSKEY macros or batch files that fill in switches and the form factor, etc., if you have only one size and

kind of floppy drive and always format for the same capacity. As soon as you start mixing sizes or format capacities, however, you're back to punching in at least the numbers manually.

The DOSSHELL doesn't really help here, either. All it gives you is a dialog box. You still have to fill in any switches and the format capacity you want (if not the defaults), just the same as on the command line. And for what it doesn't do, it adds maybe ten seconds to the task, depending on the speed of your machine. I hate to have to say it, but when it comes to FORMAT, you're just going to have to bite the bullet. Fortunately, it's not as hard to chew on as it used to be.

GOTO

Internal to COMMAND.COM

Function Instructs batch file to jump to labeled subroutine.

Usage Intended exclusively for use in batch files.

Origin Version 2.x.

Format GOTO *label*

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

This is not only one of MS-DOS's most powerful batch file tools, but also one of the easiest to use. Simple and direct, it's as valid in today's DOS environment as when it first came on the scene.

While GOTO can be used as a stand-alone command, it's most useful when used in conjunction with a conditional that tests for some condition and then either jumps or doesn't jump to a different routine depending on the outcome of the test. Here it is, for example, in a batch file that runs every time I boot my system:

```
if !%TODAY%= =!%4 goto END
Vopt c:
Vopt d:
copy lastdate.bat e:date
:END
```

If the current date matches the date of the last time my hard disk was unfragmented, the batch file skips to :END. If the dates don't match, it unfragments the disk and copies the date to a RAM disk where its presence will initiate another action. GOTO is also useful when the test is IF ERRORLEVEL or IF EXIST, as shown below:

```
IF ERRORLEVEL 0 GOTO ....
IF NOT ERRORLEVEL 0 GOTO ....
IF EXIST filename GOTO .....
IF NOT EXIST filename GOTO .....
```

Almost any string can be a label so long as it doesn't contain any separators, such as colons, semicolons, or equal signs that DOS could confuse with characters that

have a special meaning. I try to pick a word or abbreviation that explains the jump. Take note though. On the line that says GOTO *label*, you don't precede the label with a colon. You must have a colon, however, before the label (:*label*) at the beginning of the labeled line, as in the :END example on the previous page.

About the only drawback to using GOTO is that it can, in certain circumstances, lead to very long, unwieldy batch files. In those cases, I prefer to use CALL instead of GOTO to execute a separate, shorter batch file if the test is positive. Version 4.x of DOS added CALL to DOS's batch file bag of tricks, and it, after executing the called function, returns control to the calling batch file. For many purposes however—and especially in short batch files—GOTO still is the best way to get there.

GRAFTABL

External (.COM)

Function Loads the high-ASCII character table—usable only by CGA displays in graphics mode.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.0 and later.

Format GRAFTABL [*switch or argument*]

Switches

/? Displays on-screen help.

/STA Returns number of code page currently installed if any

Arguments

xxx Number of code page corresponding to the table wanted.

? Returns active code page number, also other pages available. Version 4.0 and later only.

Errorlevel

0 Code page successfully installed—no code page previously installed.

1 Code page was previously installed but new one successfully installed.

2 File error: no new code page installed.

3 Invalid parameter specified.

4 Incorrect DOS version.

Network use Not supported.

Normally, when CGA color adapters are in graphics mode, they can't display the high-ASCII characters (128−255). These include, among other sometimes needed characters, a number of accented foreign-language characters.

GRAFTABL allows users to specify and load a table of graphics characters specific to a code page. These tables contained descriptions of these high-ASCII characters.

This is one of those commands that does what it's supposed to do for certain user groups. However, this utility is specifically intended for use only with CGA adapters. Even there, it isn't needed with PC-Jrs. Also, EGA and most other graphics adapters contain bit map tables for the extended character set in the video adapter board's ROM BIOS. (Not to be confused with the system ROM BIOS discussed elsewhere in this book, EGA cards and many other video adapters contain their own ROM BIOS, similar in function to the system BIOS, but serving only the I/O needs of the video adapter.)

This, combined with the fact that many users have no call to use the high-ASCII characters—and often not in graphics mode if any exists at all—puts this one in the category of excess baggage on most hard disks.

GRAPHICS

External (.COM)
Significantly enhanced

Function Allows MS-DOS to send the contents of graphics screen display to supported graphics printers.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format GRAPHICS [*type*][*profile*][*switch(es)*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/r Reverses black/white values.

/b Valid with printer parameters COLOR4 and COLOR8, and prints background in color.

/Lcd Prints image using LCD rather than CGA screen aspect ratio.

/printbox:*id* Selects LCD or STD (CRT) display modes to emulate (v 4.0 and later).

type Printer protocols currently supported are:

COLOR1 IBM PC color printer with black ribbon.
COLOR4 IBM PC color printer with RGB (4 color) ribbon.
COLOR8 IBM PC color printer with CMY (4 color) ribbon.
DEFAULT Any Hewlett-Packard PCC printer.
DESKJET Hewlett-Packard DeskJet printer.
GRAPHICS IBM Personal Graphics, ProPrinter, or QuietWriter.
GRAPHICSWIDE Wide carriage IBM Personal Graphics Printer.
LASERJETII Hewlett-Packard LaserJet II printer.
PAINTJET Hewlett-Packard PaintJet printer.
QUIETJET Hewlett-Packard QuietJet printer.
QUIETJET PLUS Hewlett-Packard QuietJet Plus printer.
RUGGED WRITER Hewlett-Packard ThinkJet printer.
RUGGED WRITER WIDE Hewlett-Packard Rugged WideWriter printer.
THERMAL IBM PC-convertible thermal printer.
THINKJET Hewlett-Packard ThinkJet printer.

profile Specifies the name of the file containing information on all supported printers. If omitted, MS-DOS looks for file GRAPHICS.PRO in the current directory.

Errorlevel Not supported.

Network use Supported.

A major upgrade! A number of the illustrations in this book are from prints made using GRAPHICS—the first time I had been able to do that on a LaserJet-compatible printer other than by using sophisticated 3rd-party software that costs more than the whole DOS 5.0 package. It would have been possible under 4.x and possibly earlier, but only with the expenditure of considerable time to write a suitable support file for my printer.

Users are sometimes confused by GRAPHICS—when they need to use it and when plain old PrtScr would do the job. If the computer is operating in graphics mode, then the printer, too, must be in graphics mode to emulate it.

Obviously, then, any screen art, whether freehand line art or delicately shaded (including desktop publishing or the output of any graphics program such as AutoCAD), requires that the printer output be bit-mapped and printed as graphics. Windows runs in graphics mode, as do all Windows-specific applications. Most applications that run in graphics mode either have their own GRAPHICS-like printer drivers that support popular printer types. GRAPHICS fills an important niche, however, and the significant way in which it has been enhanced in version 5.0 is certainly most welcome.

So, getting back to the confusion, how do you know if you need to use GRAPHICS? You'll know. If your printer suddenly goes berserk with some program, but works just fine with all the rest, and if you have any reason to suspect you might be running your computer in graphics mode, and if your printer is similar to any of the supported types, try loading GRAPHICS and specifying your printer type first. Then try it again and see if it doesn't solve the problem.

HELP

External (.EXE)
NEW!

Function Provides on-screen help for supported DOS commands (external commands plus those supported internally by COMMAND.COM).

Usage Can be executed from the DOS command line.

Origin Version 5.0.

Format HELP *command*
or *command* /? (this usage doesn't require HELP.EXE)

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Supported.

In a move long overdue, DOS has finally added on-screen help, beginning with version 5.0. Though hardly over-generous, it should at least serve as a refresher for anyone who is reasonably familiar with DOS, and save many trips to the documentation. Shown in Fig. 20-10, I've shown the help that's provided when you type xcopy /? and Enter.

The HELP utility, in essence, simply executes the program—in this case, XCOPY—tacking the /? switch after it. The actual HELP, if available, is intrinsic to the DOS (or other) program itself, as shown in an excerpt from a dump of XCOPY.EXE, shown in Fig. 20-11.

```
Copies files and directory trees.

XCOPY pathname1 [pathname2] [[/A]|[/M]] [/D:date] [/S [/E]] [/P] [/W] [/V]

  pathname1  File(s) to copy.
  pathname2  Location and/or name of new files.
  /A         Copies files with the archive attribute set,
              doesn't change the attribute.
  /M         Copies files with the archive attribute set,
              clears the archive attribute.
  /D:date    Copies files modified on or after the specified date.
  /S         Copies subdirectories, except empty ones.
  /E         Copies empty subdirectories along with occupied ones.
  /P         Prompts before creating each new file.
  /W         Waits for you to press a key before copying.
  /V         Verifies each new file.
```

20-10 This help screen for XCOPY is typical of new on-screen help available for all MS-DOS 5.0 utilities and commands internal to COMMAND.COM.

```
001080     Copies files and directory trees.NXCOPY pathname1 [path
0010C0   name2] [[/A]|[/M]] [/D:date] [/S [/E]] [/P] [/W] [/V]  path
001100   name1  File(s) to copy.1  pathname2  Location and/or name of n
001140   ew files.;  /A Copies files with the archive attribute s
001180   et,-/ doesn't change the attribute.;  /M$ Copies f
0011C0   iles with the archive attribute set,-/ clears the archive
001200    attribute.D  /D:date    Copies files modified on or after the
001240    specified date.8  /Si Copies subdirectories, except empt
001280   y ones.D  /E0 Copies empty subdirectories along with occu
0012C0   pied ones.5  /P< Prompts before creating each new file.
001300   ;  /W- Waits for you to press a key before copying.&  /V3
001340    Verifies each new file.
```

20-11 A look inside a DOS 5.0 utility reveals text for help that will be displayed for that command—although not formatted as seen on screen.

When executed, the HELP utility neither adds to nor alters the HELP that's intrinsic to programs that support the /? switch and, in fact, the help available from any such program can be accessed without the assistance of this or any other HELP facility by simply using the /? switch.

When entered without specifying any program, however, the Help utility also works in conjunction with an ASCII file named DOSHELP.HLP to provide a general overview of available commands and their functions. This file can be modified to add unsupported DOS or nonDOS commands, although I found the documentation of this feature to be somewhat misleading because no help provided by this ASCII file can be accessed in conjunction with any specific program.

The DOSHELP.HLP file should, in fact, be considered as simply an ASCII text file that HELP displays one screenful at a time (with however many commands displayed simultaneously as the screen will hold) out of which you might find some information of limited use. As shown in Fig. 20-12, any extra lines needed for a command description should be preceded by white space. And, as a practical matter, new commands must be added in alphabetical order.

Lines beginning with @ in the DOSHELP.HLP file are comments and are ignored when HELP is executed. Otherwise, the HELP command entered by

```
@ for a command description should be preceded by white space.  Commands
@ must be added in alphabetical order.
@
@ Copyright 1990 Microsoft Corp.  All rights reserved.
@
APPEND   Allows applications to open data files in specified directories as
         if they were in the current directory.
ASSIGN   Assigns a different drive letter to an existing drive.
ATTRIB   Displays or changes file attributes.
```

20-12 Excerpt from DOSHELP.HLP file showing how it's formatted. Lines preceded by the symbol @ are remarks and won't be displayed.

itself displays the entire contents of the DOSHELP.HLP file one screenful at a time in much the same way as if it was input to MORE (MORE < DOSHELP.HLP).

It should also be noted that HELP.EXE plays no role in displaying the extensive HELP specific to DOSSHELL, QBASIC, or the EDIT function of QBASIC.

HIMEM.SYS

External (device driver)

Enhanced

Function Manages the use of extended memory.

Usage Can be installed only from the CONFIG.SYS and *must* be installed ahead of any applications or other device drivers that use extended or expanded memory or the high memory area (HMA).

Origin Version 4.0.

Format DEVICE = [*d:*][*path*]HIMEM.SYS [/hmamin = *mm*][/numhandles = *nnn*] [/int15 = *xxxx*][/machine:*yyyy*][a20control:on/off][shadowram:on/off]

Switches

/hmamin = *nn* Specifies the minimum amount of memory (0−63K) an application must use before allowing HMA access. The default is 0.

/numhandles = *mmm* Specifies the number of extended memory block (EMB) handles (1−128) that can be used simultaneously. The default is 32.

/int15 = *xxxx* Allocates a specific amount of extended memory (K) for INT15 interface. The value of *xxxx* should be at least 64K.

/machine:*yyyy* Specifies the a20 handler to be used.

The following values are currently supported:

1 or at (for IBM PC/AT)
2 or ps2 (for IBM PS/2)
3 or pt1cascade (for Phoenix Cascade BIOS)
4 or hpvectra (for HP "Classic" Vectra, or A&AT
5 or att6300plus (for ATT 6300 plus)
6 or acer1100 (for Acer 1100)
7 or toshiba (for Toshiba 1600 and 1200XE)
8 or wyse (for Wyse 12.5 MHz 286 m/c)

/a20control:on/off If set on, HIMEM will take control if the a20 line was on when HIMEM was loaded—if set to off, it will take control only if a20 line was off when HIMEM was loaded.

/shadowram:on/off If set off, HIMEM will try to switch off shadow RAM and use that RAM as part of its memory pool.

Errorlevel Not supported.

Network use Not supported.

This is the key to any use of memory beyond the nominal 640K of conventional memory with DOS. Applicable only with 80286 and higher systems, unless HIMEM.SYS (or equivalent 3rd-party driver) is loaded, there *is* no memory beyond 640K—a bunch of high- priced chips perhaps, but no usable memory. There are actually three kinds of memory that can be made available beyond 640K: extended memory, expanded memory, and an area, the HMA, that's contiguous to extended memory and that can either be a part of it or treated separately.

Actually, to refer to these as different "kinds" of memory is somewhat off the mark because the actual memory used by any or all computer functions is drawn from a common memory pool in much the way you might draw one cup of water to make coffee, another to make tea, and yet another lemonade.

Each of these memory areas requires its own unique form of memory management, however, so in order to have and use all three, you need to have three kinds of memory management software installed. HIMEM.SYS provides two of the three. It manages extended memory and then allows access to a portion of that—the 64K immediately adjacent to DOS (1024K to 1088K)—which is generally referred to as the High Memory Area (HMA). It's by using this High Memory Area (which, unlike extended memory, is accessible to DOS) that DOS 5.0 is able to relocate all but about 17K of itself—some 47K—out of conventional memory, freeing that space for use by memory-hungry applications, TSRs, etc.

While most programs using extended memory today conform to the Lotus/Intel/Microsoft/AST Extended Memory Specification (XMS), there are still some that don't. Nonconforming programs are generally not compatible with memory managers written to this specification. However, HIMEM.SYS makes allowance for such applications by allowing you to set aside some portion of extended memory. This is done by using the /int15 switch on the HIMEM command line in the CONFIG.SYS, as shown here, which would set 1 megabyte (1024K) aside for nonconforming software while retaining the remainder for conforming applications:

```
DEVICE = \DOS\HIMEM.SYS /int15 = 1024
```

HIMEM.SYS must be installed ahead of any other drivers that use memory above 640K. This includes not only EMM386.EXE, which emulates expanded memory, but any DEVICEHIGH= commands in the CONFIG.SYS. I generally put it first, although in certain instances—computers upgraded by accelerator cards, for instance—it might be necessary to install it after the driver.

IF

Internal to COMMAND.COM

Function Performs a conditional test, used typically in batch files.

Usage Intended primarily for use in batch files.

Origin Version 2.x.

Format IF [NOT] ERRORLEVEL *number command*
or IF [NOT] EXIST *condition command*
or IF [NOT] *string1* = = *string2 command*

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Can use ERRORLEVEL as a conditional test.

Network use Not supported.

IF I were a king, I couldn't ask for a handier, more versatile, and more useful MS-DOS command than IF. IF and its companion IF NOT can be surprisingly powerful, using some condition as a test to determine the following course of action. While often the choice of which command to use is obvious, in many cases they can be used interchangeably—at least in terms of doing whatever it is you're trying to accomplish. Usually one will make the job a little easier or more direct than the other, however, and I'll frequently write a batch file one way, look at it, see a better way to do it, and rewrite it.

To understand the workings of IF, let's take an example from one of my actual batch files and dissect it, word by word. The various parts of the following line:

```
IF !%TODAY%= =!%4 GOTO END
```

mean the following:

IF	If
!%TODAY%	(a variable)
= =	is the same as
!%4	the 4th input parameter
GOTO	skip to
END	(a label)

TODAY—the date—is a variable I use regularly in my environment, as described in the chapter where I deal with batch files in greater depth. END is a label which, appropriately, is the last line of the batch file. If the 4th parameter passed to the batch when it executes is a date that matches the date previously set in the environment, then it skips past any other lines below this one—one, ten, or a hun-

dred, it doesn't matter—until it finds the :END label. It executes those other lines only when the dates don't match—which means that, no matter how many times I crash my system and have to reboot during the day (and in this business that happens quite a lot), whatever those other lines are supposed to do gets done only once each day. The following:

```
IF NOT ERRORLEVEL n GOTO label
IF ERRORLEVEL n PAUSE|ECHO message
.
.
```

is another good—and fairly typical—use of the IF command. In this case, I've used it twice. If the errorlevel passes, the batch file jumps directly to the label. If not, it's tested again and one of the remaining possibilities causes the batch to pause and prompt with the *message* before proceeding—which also gives you the opportunity of breaking out (^C) and terminating the batch job at that point. If, in the second case, the errorlevel doesn't match the one specified, the batch job doesn't pause or echo the message, but simply continues.

So if it's so useful, how often do I use it? I was curious too, so out of curiosity I copied my entire batch file directory into a single file and checked it with FIND:

```
find "if" e:bat /c/i
---------- E:BAT: 82
```

Eighty two times in one directory—and I've got batch files other places too. If I checked my entire hard disk . . . IF . . .

INSTALL

Internal to IO.SYS

Function Loads terminate-and-stay-resident (TSR) programs from CONFIG.SYS rather than from command line or batch file.

Usage Can be executed only from the CONFIG.SYS.

Origin Version 4.0.

Format INSTALL = [*d:*][*path*]*filename.EXE* [*parameters*]

Switches None.

Errorlevel Not supported.

Network use Not supported.

A newer command, the purpose of INSTALL is to load certain DOS TSRs from the CONFIG.SYS. This, as documented by Microsoft when INSTALL was first introduced, is of benefit because it causes supported programs to be loaded at lower addresses than would be available later.

There are only four programs officially supported by INSTALL: FASTOPEN.EXE, KEYB.EXE, NLSFUNC.EXE, and SHARE.EXE—and at least two of these are used only by a relatively small group. INSTALL is being used by some 3rd-party software developers—one security software developer, in particular, is using it to enhance the security of his software—but Microsoft officially frowns on any such use.

The validity of INSTALL for its intended purpose seems subject to question in 5.0 as it, in fact, runs contrary to Microsoft's efforts to allow us to move as much of DOS as possible entirely *out* of conventional memory. So I would check the other options out rather carefully before I put any of my eggs in this basket.

IO.SYS

Hidden system file

Function Primarily intended to enhance and extend the I/O capability provided during bootup by the ROM BIOS.

Usage Can be loaded only by system during bootup.

Origin Version 1.0. Only the PC-DOS equivalent actually dates to version 1.0. MS-DOS didn't emerge until later when the clone market and the need for a generic rather than device-specific DOS began to develop.

Format N/A.

IO.SYS in MS-DOS is functionally similar to IBMBIO.COM in PC-DOS. At the system level, the .SYS and .COM extensions can be used interchangeably.

Together with it's companion MSDOS.SYS, these two carefully protected files, invisible to the casual observer even in ordinary directory listings, are the very heart and soul of DOS. Without them you don't have DOS, and without both intact and loaded properly a machine won't finish booting as a DOS machine.

These are the only two files that cannot be replaced by functional equivalents from 3rd-party sources without, in fact, running under some operating system other than MS-DOS. The I/O capabilities provided can be further enhanced to include support for new devices not anticipated at the time of its release or support for nonstandard devices by means of device drivers that provide system interfaces specific to those needs.

Some idea of the role of IO.SYS can be gained from an examination of the internal commands supported by it, even in the absence of COMMAND.COM as3, shown here:

BREAK	DEVICE	FCBS	REM
BUFFERS	DEVICEHIGH	FILES	SHELL
COMMENT	DOS	INSTALL	STACKS
COUNTRY	DRIVPARM	LASTDRIVE	SWITCHES

Note that none of the above can be entered from the DOS command line, but only from the CONFIG.SYS, which is read and executed before COMMAND.COM (or any alternative command interpreter) is loaded. Error messages specific to I/O functions are also generated directly by IO.SYS.

Note also that HELP is not provided for any of these either because, by the time HELP is available, access to any of them is denied. Changes in the use of any of them—the addition or deletion of some functions or any changes in parame-

ters—can be accomplished only by editing the CONFIG.SYS file, and then those changes won't take effect until the system is rebooted.

Because IO.SYS and MSDOS.SYS (or the PC-DOS equivalents) are so crucial to system operation, the R, S, and H attributes are all set during installation to prevent inadvertent alteration or erasure. Also, because they're loaded before the system is fully operational and are in fact required to bring it to a fully operational level, they must reside at specific locations on any disk for that disk to be bootable.

JOIN

External (.EXE)

Function Links a physical or logical drive to a subdirectory on another drive.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.0 and later.

Format JOIN *d1: d2:\path* (to join a drive to an empty directory)
or JOIN (to display the current status of JOIN)
or JOIN *d1:* /d (to cancel a current JOIN)

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

While writing this book, I went through every command in 5.0 rather carefully. But try as I might, I can't think of one redeeming feature to recommend this command for use in today's DOS environment—although JOIN, as you can see, isn't a prehistoric dinosaur dating back to DOS's earliest beginnings.

At first glance, JOIN seems similar to APPEND, but is in fact entirely different—a bit more like ASSIGN, but not exactly. JOIN can be used only to join one disk drive to an *empty* subdirectory on a different drive—say on your hard disk. All it allows you to do is swap floppies with data for your application(s) in and out.

Any files or data that are written to joined disks are written to the extrinsic disk. Nothing is ever written to the subdirectory it's joined to, which must remain completely empty. In other words, the subdirectory is really nothing but a middleman, an access channel. Yet the joined disk drive can't be accessed directly while JOIN is in effect.

On the plus side, JOIN is not a TSR (like APPEND, for example) so it doesn't tie up a block of memory. And, although at one time JOIN apparently suffered from being unable to see beyond the first level of subdirectories on the joined disk, in this release I chased down and actually accessed files through no less than four directory levels.

Curiously, although the DOSSHELL file manager had no trouble tracking files through the various subdirectory levels on a joined disk, XTREE Pro Gold showed the directory I had joined to was completely empty. I also found JOIN to be unworkable from within DESQview. This is a strange one.

JOIN has a number of documented limitations, as well. You shouldn't use it in combination or conjunction with SUBST or ASSIGN. Nor should BACKUP, RESTORE, FORMAT, DISKCOPY, or DISKCOMP be used with JOIN.

KEYB[.COM]

External (.COM)

Function Loads keyboard device drivers that support other than U.S. keyboards.

Usage Can be executed from the DOS command line, in batch files, or from the CONFIG.SYS using the INSTALL= function.

Origin Version 3.0 and later.

Format KEYB [*argument(s)*][,[*d:*][*path*]*filename*[*.ext*]][*switch*]
or KEYB (entered without parameters to display current code page)
or INSTALL=KEYB.COM [*argument(s)*][,[*d:*][*path*]*filename*[*.ext*]][*switch*]

Switches

/e Directs KEYB to assume that enhanced keyboard is installed—useful mainly with 8086/8088 machines.

/id:zzz Used to specify keyboard code from countries having more than one keyboard layout. DOS 4.x only.

Arguments

xx Nationality.

yyy Keyboard program to load.

Errorlevel

0 Keyboard driver successfully installed.
1 Improper keyboard code number, code page number, or syntax.
2 Bad keyboard definition file or file not found.
3 Could not load driver into memory.
4 Error with CON device.
5 Code page requested has not been prepared.

This program replaces the earlier KEYB*xx* program in versions 3.0 through 3.2, which was DOS's first attempt to add support-installable keyboard device drivers for other than the default standard U.S. keyboard layout.

KEYB is just one of several DOS utilities specific to multilingual usage, and as such is of interest to only a relatively small user base, because the default for MS-DOS sold in this country is the us (United States) keyboard. For those who do require it, the documentation seems sufficiently self-explanatory to require no elaboration here. You should note, however, that KEYB is one of the four DOS utilities that can be installed either from the command line or optionally from the CONFIG.SYS using the INSTALL= command.

LABEL

External (.EXE)

Function Adds or changes the disk volume label.

Usage Usable mainly from the DOS command line.

Origin Version 3.0 and later.

Format LABEL [*d:*][*name*]

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Not supported.

While you can also put volume labels on your disk while formatting, this I find is usually the handier way because, when I buy disks, I generally just go ahead and format the whole box, usually with no idea at that stage what I'm going to use them for.

There really isn't much that you can do wrong with LABEL. And, because it's one of DOS's friendlier commands, it even prompts you, telling you how many characters you can use:

```
Volume in drive B has no label
Volume Serial Number is 15C9-3F43
Volume label (11 characters, ENTER for none)?
```

You can't use colons, semicolons, periods, slashes, etc., as part of label names or it will come back and tell you:

```
Invalid characters in volume label
```

You can, however, use hyphens or the underline character to connect words, as in test_disk_1—just 11 characters, count them. You can also use LABEL to rename a disk. Or you can simply delete a volume label if the disk you tell it you want to label already has a name, by just hitting the Enter key, at which point DOS will prompt you:

```
Delete current volume label (Y/N)?
```

LABEL—like so many other commands—shouldn't be used in conjunction with ASSIGN or SUBST, but it's unlikely you'd ever have occasion to try. A really handy little utility, it's just too bad that everything else in DOS isn't this easy and foolproof—but then I'd be out of a job, wouldn't I . . .

LASTDRIVE

Internal to IO.SYS

Function Sets the maximum number of physical or logical drives that can be accessed.

Usage Can be used only in the CONFIG.SYS.

Origin Version 3.0.

Format LASTDRIVE = c (any letter A – Z)

Switches None.

Errorlevel Not supported.

Network use Supported.

There's an error in at least the early 5.0 MS-DOS documentation regarding the use of this command. It states that *This command is useful only in a network environment*. This is completely false. Certainly, it's necessary to use LASTDRIVE to specify something other than DOS's five-drive default in most networking situations, as it's equally essential in conjunction with the SUBST command. And SUBST is, in my opinion, one of MS-DOS's most useful utilities—I don't know what I'd do without it.

Because the use of LASTDRIVE in connection with network operations is usually well documented by the network software manuals, I won't belabor the point here—and anyway, the command functions exactly the same, whether you're networking or using SUBST.

Installable only from the CONFIG.SYS, LASTDRIVE does use up some memory. It takes roughly 80 bytes for each additional drive you provide for, 1840 bytes to go the limit (Z)—and working from the CONFIG.SYS you have to make allowances at boot time whether you actually need all of the drives you anticipate or not. Some 3rd-party memory managers supply LASTDRIVE-type programs that allow you to use upper or what Microsoft refers to as "reserved" rather than conventional memory for this, but even though version 5.0 allows you to relocate most of DOS on 286 and higher machines, it doesn't support this. Even in conventional memory, however, the added overhead seems a small price to pay for the convenience—if not the necessity—of adding more drives, even just pretend drives, to your system.

LOADFIX

External (.COM)

NEW!

Function Used to load programs that have difficulty loading in newly freed low conventional memory—particularly below 64K.

Usage Used from command line or batch files.

Origin Version 5.0.

Format LOADFIX [*d:*][*path*]*program.ext*

Switches None.

Errorlevel Not supported.

Network use Supported.

With DOS 5.0 able to load so much of itself plus most device drivers above 640K on 386 and higher systems and having a lot of additional conventional memory to work with, it now seems that some programs have a problem loading, particularly when DOS tries to load them below 64K. To cope with this problem, Microsoft slipped LOADFIX in near the end of the DOS 5.0 beta program.

The error message *Packed file corrupt* is symptomatic of this condition and, while it was observed a few times during the preparation of this book prior to the addition of LOADFIX, efforts to recreate the problem to verify the fix have been inconclusive at this point.

Nonetheless, users of 386 and higher systems, particularly those using HIMEM.SYS or other XMS managers to load the bulk of the DOS kernel to the High Memory Area should be aware of this new utility and its purpose.

LOADHIGH

Internal to COMMAND.COM
NEW!

Function Used to load a program into upper (reserved) memory blocks.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 5.0.

Format LOADHIGH [*d:*][*path*]*filename* [*parameter(s)*]
or LH [*d:*][*path*]*filename* [*parameter(s)*]

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Not supported.

Playing an important part in MS-DOS's new memory management scheme, LOADHIGH can result in significant gains in the amount of conventional memory left to run applications on 386 and higher machines. However, it's also certain to cause many users trouble. I wish there was some easy answer, but any time you start poking around addresses above 640K you can expect some problems. Some of the glossier 3rd-party memory management packages provide special proprietary utilities that make life easier, while trying to determine what will and will not load to—and run from—upper memory addresses successfully. Unfortunately, DOS doesn't. So while LOADHIGH can be used successfully to relocate TSRs up there, it's a trial-and-error process.

As in loading programs normally, each one you want to load in upper memory requires a separate command line beginning with LOADHIGH, as shown here:

```
LOADHIGH MODE COM1:9600,N,8,1,P
LOADHIGH MODE COM2:9600,N,8,1,P
LOADHIGH XYWRITE \XYKBD
```

Here I've loaded MODE in upper memory and initialized two serial ports, resulting in not a huge gain—but every little bit helps. And just to prove a point, I've thrown in a TSR that belongs to my word processor and used LOADHIGH to kick it into upper memory, too.

Typically, a 386 will have as much as 128K of recoverable address space above 640K. Obviously, you can't recover all of it, but you can use a good deal of it. And every kilobyte of TSRs you relocate up there gives you one more kilobyte

of conventional memory to run your applications—or to run some TSR you've never been able to afford the memory for before. I load a lot of things above 640K on my machines and, believe me, it's well worth the effort. Even though it generally takes some trial and error to achieve the best results.

MEM

External (.EXE)
Enhanced!

Function Displays information on memory utilization.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 4.0 and later.

Format MEM [*switch*]
or MEM (entered without any switch displays the current status)

Switches

/? Displays on-screen help including syntax, switches, etc.
/classify (or /c) Displays program information in decimal form.
/debug (or /d) Similar to above except it includes system device drivers.
/program (or /p) Specifies programs, device drivers loaded in memory, etc.

Errorlevel Not supported.

Network use Supported.

The only totally new external command added by version 4.0, MEM has come into its own only with DOS 5.0. Even here, however, MEM is of limited use except to power users determined to squeeze the maximum performance out of DOS 5's HIMEM.SYS memory manager.

Still, even for the casual DOS user, MEM provides some very useful information. Entered without any of its switches, MEM returns something like what's shown in Fig. 20-13. Look at the chunk of memory left over for applications with the DOS 5.0 kernel kicked upstairs to the High Memory Area—627216—over *627*K!

```
655360 bytes total conventional memory
655360 bytes available to MS-DOS
627216 largest executable program size

458752 bytes total EMS memory
 65536 bytes free EMS memory

3407872 bytes total contiguous extended memory
      0 bytes available contiguous extended memory
1986560 bytes available XMS memory
        MS-DOS resident in High Memory Area
```

20-13 MEM report showing over 627K of usable conventional memory available with bulk of MS-DOS 5.0 kernel moved to HMA.

That's just the raw data, however. MEM's three switches also display this data, but return a lot of detailed memory usage data, as well—useful information about what's loaded where. Prior to 5.0, the biggest problem was that, other than the raw data when MEM got down to business, everything was hexidecimal only.

A few things happened in this release that elevate MEM to a position of importance, particularly in the world of 386 and higher machines:

- MEM learned about decimal numbers.
- HIMEM.SYS, with it's EMS companion manager EMM386.EXE, can map memory and load DOS programs—device drivers and TSRs—up above 640K, where DOS has never been before,
- *Only* MEM can show you just what HIMEM.SYS and EMM386.EXE are really doing for you up there and how much more space—if any—you still have available.

Unfortunately, the new /c switch is still the only one that does decimal—and that's doubly unfortunate because it *only* does decimal (with the exception of duplicating the file size information, as shown in the dump in Fig. 20-14). And, if you want to get down to the nitty-gritty, there is some hex information—particularly address information the /c switch doesn't give you any which way—that you're going to want.

To get the total picture, then, you might want to run MEM /c and then MEM /d, if only to see what addresses you're dealing with. Particularly if you start running out of memory upstairs and have to start devising clever load sequences that pack in more goodies. Figure 20-15 shows an excerpt from a MEM /d where there are specific address locations you can match up with the individual TSRs and device drivers. You would need that information if you had to start shuffling the deck.

Still, just the /c switch is adequate to get most users up and running anyway—and is probably all most users will need or even want. Just keep in mind the other two are there.

The /debug switch, by the way, has nothing to do with the DEBUG program, so even if you're squeamish about using DEBUG you shouldn't hesitate to use the /debug (or /d) switch if your curiosity gets the better of you. You'll want to either use the filter MORE to keep the several screenfuls you're going to get from scrolling by too fast for any good use, or else redirect it to a file you can browse through at leisure with your word processor or editor. The commands entries for these usages are:

MEM /DEBUG|MORE (to view it on screen)
MEM /DEBUG >[*d:*][*path*]*filename* (to redirect it to a file)

```
Conventional Memory :

  Name                 Size in Decimal          Size in Hex
------------      ----------------------   ------------
  MSDOS                 15536      ( 15.2K)      3CB0
  HIMEM                  1184      (  1.2K)       4A0
  EMM386                 8400      (  8.2K)      20D0
  COMMAND                2624      (  2.6K)       A40
  FREE                     64      (  0.1K)        40
  FREE                 627344      (612.6K)     99290

Total  FREE :          627408      (612.7K)


Upper Memory :

  Name                 Size in Decimal          Size in Hex
------------      ----------------------   ------------
  SYSTEM               184320      (180.0K)     2D000
  RAMDRIVE               1184      (  1.2K)       4A0
  SMARTDRV              17904      ( 17.5K)      45F0
  EGA                    3280      (  3.2K)       CD0
  STACKER               22336      ( 21.8K)      5740
  MODE                    464      (  0.5K)       1D0
  XYKBD                  1552      (  1.5K)       610
  FREE                     48      (  0.0K)        30
  FREE                 30880      ( 30.2K)      78A0

Total  FREE :          30928      ( 30.2K)

Total bytes available to programs (Conventional+Upper) :  658336   (642.9K)
Largest executable program size :                         627216   (612.5K)
Largest available upper memory block :                     30880   ( 30.2K)

  458752 bytes total EMS memory
   65536 bytes free EMS memory

 3407872 bytes total contiguous extended memory
       0 bytes available contiguous extended memory
 1986560 bytes available XMS memory
         MS-DOS resident in High Memory Area
```

20-14 MEM report on memory usage using the new /c (classify) switch.

Again, a reminder: MEM can read what's going on upstairs *only* if you're using HIMEM.SYS and EMM386.EXE. When 3rd-party memory managers like QEMM or 386MAX are loaded, MEM can only show what's loaded in conventional memory.

```
01E4D0          MSDOS           081B10      -- Free --
09FFF0          SYSTEM          02D010      System Program

0CD010          IO              00AEE0      System Data
                  RAMDRIVE      0004A0       DEVICE=
                    E:                        Installed Device Driver
                  SMARTDRV      0045F0       DEVICE=
                    SMARTAAR                  Installed Device Driver
                  EGA           000CD0       DEVICE=
                    EGA$                      Installed Device Driver
                  STACKER       005740       DEVICE=
                    F:                        Installed Device Driver
0D7F00          MSDOS           000030      -- Free --
0D7F40          MODE            0001D0      Program
0D8120          XYKBD           000040      Environment
0D8170          XYKBD           0005D0      Program
0D8750          MSDOS           0078A0      -- Free --
```

20-15 Excerpt from a MEM report with /d (debug) switch.

MIRROR

External

NEW!

Function Records data about one or more disks that allows REBUILD or UNDELETE to recover erased files or restore disk after accidental reformatting.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 5.0.

Format MIRROR [*d:*][*switches*]
or MIRROR /PARTN
or MIRROR /U

Switches

/? Displays on-screen help including syntax, switches, etc.

/I Saves only the most recent copy of MIRROR.FIL (the default saves last previous, if any, as MIRROR.BAK).

/t*drive*[-*entries*] Creates tracking file for use by UNDELETE in recovering erased files. Optional entries specifies maximum number.

/PARTN Saves hard disk partitioning data for use by REBUILD to restore hard disk with corrupted partition data.

/U Unloads delete tracking (/t...) from memory. Effective only when this program loaded last.

Errorlevel Not supported.

Network use Supported.

Disk management has been more enhanced in version 5.0 than in any previous single release. MIRROR, in conjunction with DOS's new nondestructive format capability, provides the means for recovering erased file or, in many cases, restoring an entire hard disk (or floppy) after an accidental reformat or catastrophic disk crash.

MIRROR plays the key role in data/disk recovery by creating image files as backups for the most critical areas on your disks: the partition table on your hard disk, the FAT (File Allocation Table), and the root directory of your hard disk (or floppies).

Because this is the information MIRROR's companion program UNFOR MAT uses when attempting to restore a disk after an accidental reformat or other recoverable occurrence, MIRROR should be run frequently to keep the file MIR ROR.FIL as nearly up-to-the-minute as possible.

Unlike a complete hard disk backup that might take some minutes to complete, it typically takes MIRROR only a few seconds to update its backup of the FAT and root directory. I clocked it at just under 5.16 seconds to update its copy of the systems data—50,688 bytes—for a 33Mb partition on a 16Mhz machine. Because it requires so little time, MIRROR is something that can easily be included in your AUTOEXEC.BAT so it's done automatically every time you boot your system, limiting the potential risk of loss to files created or modified during the current work session.

By default, MIRROR saves the previous MIRROR.FIL (if any) as MIRROR .BAK whenever it writes a new, updated file. Should the current MIRROR.FIL become corrupted, this would hopefully give you a second shot at recovering files up to the point covered by that earlier data. There are arguments pro and con on this issue. If the /l switch is used, MIRROR ends the argument by saving only the current data.

Users who regularly unfragment their hard disk *must*, however, rerun MIRROR as soon as possible after any and every unfragmenting or disk optimizing operation, and should also erase any MIRROR.BAK file at that time. The FAT and root directory entries are a road map to guide DOS back to all the often widely scattered bits and pieces of your files. Unfragmenting collects those scattered bits pieces and moves them somewhere else, and the last thing DOS needs is a map that knows only where things used to be.

Tracking and recovering erased files

In addition to the protection MIRROR offers against accidental reformatting of your hard disk or other catastrophic occurrences, it also has an optional feature that keeps track of deleted files to facilitate their recovery by MIRROR's companion program UNDELETE—also new to DOS in this release. This is accomplished through a tracking file, PCTRACKR.DEL, that's created or updated by MIRROR whenever files are deleted and when it's active (memory resident).

Note that the use of MIRROR only facilitates the job of unerasing files. UNDELETE also functions as a completely independent and free-standing file recovery utility, so the use of MIRROR with /t*d:*[-*entries*] switch isn't mandatory. See the UNDELETE command for the details regarding its use independent of MIRROR. It does make it easier, especially for less experienced DOS users, because this file recovery feature virtually eliminates guesswork, as shown in Fig. 20-16.

As shown, you have not only the *entire* filename (with most undelete utilities the first letter is missing and must be filled in), but also the history and an assessment of the file's recoverability, requiring only a Y/N input to either undelete or skip on past this entry.

To use this tracking function of MIRROR, it must be loaded as a resident (TSR) program at the time the files are deleted—which, as a practical matter,

```
Directory: A:\
File Specs: *.*

    Delete Tracking file contains
    3 deleted files.
    Of those,     3 files have all clusters available,
                  0 files have some clusters available,
                  0 files have no clusters available.

    DOS Directory contains   15 deleted files.
    Of those,    15 files may be recovered.

Using the Delete Tracking file.

      ATTRIB   EXE
16223  8/15/90  3:33a  ...A  Deleted: 10/30/90  4:27p
All of the clusters for this file are available.
Do you want to recover this file? (Y/N)
```

20-16 Screen showing recovery of deleted files when MIRROR has been run in TSR mode to keep track of erasures.

pretty much means every time you boot. MIRROR, as a TSR, automatically requires 6704 bytes of RAM when you execute it with the /t*d:*[-*entries*] switch.

Note, in the above example, that UNDELETE also shows the disk contains 15 recoverable files in the DOS directory. DOS isn't the name of the directory—this refers only to the directory maintained by DOS to distinguish those files from the ones tracked by MIRROR's PCTRACKR.DEL file. These files were erased when MIRROR wasn't loaded, and therefore weren't tracked. Even so, they're probably recoverable by UNDELETE, but only if the PCTRACKR.DEL file is either deleted or renamed to something UNDELETE can't recognize.

The PCTRACKR.DEL file is always on the root directory, but contains recovery data for any file deleted from subdirectories as well. They recover deleted files from subdirectories that you must specify on the command line, as follows:

UNDELETE *d:*[*path*]

Saving hard disk partition data

A hard disk partition is nothing but a compact data file, but it's so critical that, should it become corrupted, DOS won't even recognize your disk as valid media. MIRROR used with the [*/partn*] switch copies that data to a floppy disk for safe-keeping—you have to keep your backup copy of this data some place other than on your hard disk.

It's a painless process that, done once, is valid as long as you don't alter the partition(s) on your hard disk—even with DOS's old destructive FORMAT, the partition table was left untouched. You don't have to specify which drive you want to save the data to. The command MIRROR /partn returns the message:

```
Disk Partition Table saver.
The partition information from your hard drive(s) has been read.

Next, the file PARTNSAV.FIL will be written to a floppy disk.
Please insert a formatted diskette and enter the name of the
diskette drive.
What drive? A
```

The default drive, as shown, is A, you can enter any valid drive letter to replace it. Because you'll need this disk again only in case of dire emergency, put it away in some very safe place. It could quite literally be worth its weight in gold. And then some.

I keep a partition table backup on a bootable floppy. Other than the system and COMMAND.COM, the disk contains nothing but this data file and a copy of the recovery software—in this case, REBUILD. There are, of course, other problems that can cause a hard disk to come up missing—or at least missing as far as DOS is concerned. But this is about as good as any insurance you can buy and, for the cost of a floppy and maybe two minutes of your time, the price is certainly right.

MKDIR (or MD)

Internal to COMMAND.COM

Function Creates a new directory or subdirectory.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.x.

Format MKDIR *d:*[*path*]
or MD *d:*[*path*]

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Supported.

There really isn't much that you can say about this one beyond the fact that you have to take it one step at a time. If you want to create a subdirectory called \BILLS\MONTHLY\OVERDUE and don't already have a \BILLS\MONTHLY or even a \BILLS directory, you can't just go:

```
MKDIR \BILLS\MONTHLY\OVERDUE
```

because DOS will tell you *Invalid path \BILLS\MONTHLY\OVERDUE*. So you have to take it step by step and issue the following series of commands:

```
MKDIR \BILLS
MKDIR \BILLS\MONTHLY
MKDIR \BILLS\MONTHLY\SEPTEMBER
```

The ultimate limitation on how many levels of subdirectories you can use is the length of the path. The length of any single path cannot exceed 63 characters (including backslashes) from the root directory to its lowest level. As a practical matter, the more levels DOS has to find its way through the longer it takes—because the neat three-level setup tree I've set up here is anything but neat on a disk.

Let's say you're looking for a file called OVERDUE, that's contained in the last subdirectory of the above path. To call up \BILLS\MONTHLY\SEPTEMBER\OVERDUE, DOS must first look in the root directory for BILLS. But there's nothing there but an entry pointing to file—a directory is really nothing but a file—out on the disk somewhere. So DOS goes there and finds yet another entry representing another directory file still somewhere else. So DOS goes there and only then does it find any mention of a file named SEPTEMBER—if it even exists rather than break such a complex path in your environment, you

can simplify things considerably either by specifying the entire path on the command line or, even better, having the DOS SUBST command. That takes time. FASTOPEN—or better, SMARTDRV.SYS—can help you keep it to a minimum, but even they can't help the first time.

Obviously, the ability to create an almost infinitely flexible directory structure is one of the most powerful file management tools you have, but give it serious thought before you try to take it to limit.

MODE

External (.COM)

Function Sets operating mode for parallel printer(s), sets protocols for serial ports, redirects printer output, sets operating mode for CGA adapters, prepares, activates, displays or restores code pages (v 3.3), and sets repeat rate for keys held down.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0 and later.

Format (varies with command function, see below)

Switches

/? Displays on-screen help including syntax, switches, etc. Note that other switches vary with command function.

Arguments (varies with command function, see below)

Errorlevel Not supported.

Network use Supported.

On-Screen Help None.

Wearing a lot of hats, MODE is one of the DOS TSRs. It came into being when there was little commercial software in the then infant PC market, and user needs far outstripped the little that there was. With succeeding generations of DOS, MODE has taken on new tasks and the ways in which it handles old ones has been enhanced.

However, through the years other software—notably applications software—has nibbled away at many of its functions. Most modern word processors, communications packages, and other applications and utilities that must access serial and parallel ports initialize those ports as required without MODE. While working on this book, I found a MODE command still lurking in my AUTOEXEC .BAT. The need was still there, but the function was being handled automatically by other software.

And there are many programs that, as secondary functions, can redirect printer output to other ports. Also, as CGA monitors have largely been supplanted by EGAs and VGAs, that function has taken a backseat to more contemporary issues, but it's still supported.

There are still occasions today when MODE can be quite useful, however—often enough that every user should keep it handy on his or her DOS directory.

Because the uses are rather diverse, each with its own supporting set of options, arguments, and/or switches, I've listed them here according to their special usages, rather than as single set.

To display device status

MODE [*device*][/status (or /sta)]

Mode can be used to display the status of one or all of the devices installed on your system. Entered without parameters, it gives some information about some current settings, but it's sketchy, as shown in Fig. 20-17.

```
Status for device LPT1:
-----------------------
LPT1: not rerouted
RETRY=NONE
Code page operation not supported on this device

Status for device LPT2:
-----------------------
LPT2: not rerouted

Status for device LPT3:
-----------------------
LPT3: not rerouted

Status for device CON:
----------------------
Code page operation not supported on this device

Status for device COM1:
-----------------------
RETRY=NONE
```

20-17 MODE entered without parameters displays current status of ports, etc.

Specifying a single device on the command line following MODE limits the status report to that device but, unlike some other DOS commands, adds nothing to the data displayed. The /status (or /sta) switch is required only when you're checking the current status of a *parallel* port. Failing to include this switch will cause DOS to cancel any MODE port redirection that might be in effect, rather than report the status.

To configure a parallel printer

MODE LPT*n*[:]*c*[,[*l*][,*r*]], where:

n is the parallel printer port, 1 through 3. The default is 1.
c is the number of characters per line, 80 or 132. The default is 80.
l is the lines printed per vertical inch, 6 or 8. The default is 6.

r specifies retry action to take when time-out error occurs (this function causes part of MODE to remain memory resident.

Allowable values are:

e returns error signal when printer is busy. This is the default and specifically recommended for use with IBM LANs.
b returns busy signal, replaces P in earlier versions.
r returns ready signal from port status check.
none means that no retry is attempted.

Although today this function is usually managed by word processing or other software that normally accesses printers connected to parallel ports, it's also possible to configure those printers through MODE. Though rarely needed, you should be aware of it should the need arise.

DOS 5.0 supports up to three parallel ports: LPT1, LPT2, and LPT3. LPT1 is the default (PRN can be used interchangeably with LPT1 to designate the default port).

To configure a parallel printer connected at LPT2 in order to print 132-character lines, spaced six characters to the inch, the MODE command would be:

```
MODE LPT2:132,8,b
```

The b tacked on the end returns a busy signal from a status check of the port if busy. If you're working on a network, you should avoid *r* options that initiate continuous retry attempts.

To configure a serial port

MODE COM*n*[:]b[,*p*[,*d*[,*s*][,*r*]]]], where:

n is the com port number, 1 through 4. The default is 1.
b is the data transmission rate in baud (baud/8 = characters per second). The first two digits only are required. Allowable values are:

11 (110 baud)
15 (150 baud)
30 (300 baud)
60 (600 baud)
12 (1200 baud)
24 (2400 baud)
48 (4800 baud)
96 (9600 baud)
19 (19200 baud)

p specifies the parity, where:

n is none.
e is even (the default).

o is odd.
m is a mark.
s is a space.

d specifies the number of data bits: 5, 6, 7, or 8 (the default is 7).
s specifies the number of stop bits: 1, 1.5, or 2 (the default is 1, except with 110 baud rate).
r specifies retry action to take when time-out error occurs. Allowable values are:

e returns error signal when com port is busy (default).
b returns busy if status check shows port busy.
r returns ready signal from port status check.
none means no retry is attempted.

This is probably the most familiar aspect of the MODE command to many users. It can often be the most frustrating, as well, with its obtuse terminology: baud rates, parity, stop bits, data bits, etc. And you know that if *any* of the numbers at your end don't match the numbers at the other end you're just not communicating. Whether that other end is half way around the world or right on your own desk makes little difference.

The subject of serial communication is too complex to more than barely scratch the surface here, but, for the uninitiated, I'll provide just a quick look at the meaning of the terms.

Transmission rate is fairly self-explanatory and simply means you've got to be listening at the same speed that the other guy is talking. There are some vagaries in the numbering system but, for practical purposes, if the numbers match you can communicate.

What you're communicating are bits of information, and if all you had to do was send the bits of actual data, life would be a snap. It isn't. You have to send a start bit to signal the receiving end that something's coming. Fortunately, you don't have to set a separate parameter for that, but you've got enough to do without it.

The data is most commonly sent in seven- or eight-bit bursts, and it makes a difference, because the *next* bit is the parity bit and, to computers or other communications devices, one bit looks just like the next. They're all just little voltage fluctuations. Only their positions give them different meanings. So if you're set up expecting seven bits of data followed by the parity bit, and eight come in and they all have the parity check bit one position later, it's not going to work.

Here, for example, are representations of the way a serial device sees the letter A, first with 7 data bits, then 8:

```
sddddddps
+−+++++−+−
```

```
sddddddddps
+−+++++++−++−
```

The *parity bit* (p) is a method of testing whether the transmission is being received correctly or not. The sending device adds a parity bit, which is calculated according to the contents of the data bits. The receiving device checks that bit against the incoming data, then, to verify its accuracy—not absolute but close enough for most purposes. There are five schemes for parity determination—one in which the number is always odd and another where it's always even, hence *odd* and *even* parity. Often times, there's no parity bit (or it's ignored). These are the three most commonly encountered. Just keep in mind that, if the receiving device isn't using the same error detection scheme as the sending device, you've got a problem.

Stop bits are simply a means of signaling the end point of a burst of data, called a frame. And then you start all over again.

This brief explanation is obviously too sketchy to be of much help sorting out a mismatch, but perhaps it's enough to give a little meaning to the numbers—it always seems a little easier to deal with something if you've got at least a vague idea of what that something is.

To redirect output to serial port

MODE LPT*n*[:] = COM*m*[:], where:
n is the nominal parallel printer port number.
m is the serial port to use.

I use this in the AUTOEXEC.BAT to redirect default printer output to a serial-input peripheral sharing device that handles several printers, a shared modem, etc. It's often, if not usually, necessary to precede a MODE redirection command with another MODE serial port configuration command specific that's to the baud rate and other parameters of your printer, as shown here:

```
MODE COM1:9600,N,8,1,P
MODE LPT1: = COM1:
```

Note that, although both of these are functions of the MODE command, they can't be combined into a single command, but must be treated separately. Normal parallel port output can be restored by simply entering the command without a second port specification, as shown here:

```
MODE LPT1:
```

To determine the current status of a parallel port, the /status switch must be added to the command line (see the next page) because simply specifying the device would cancel any redirection, as in the above example. The command to deter-

mine the current status of a parallel port without altering it is:

```
MODE LPT1: /status
```

Note that either the word /status or the abbreviation /sta can be used. Unfortunately, the MODE command doesn't support redirection between two parallel or two serial ports, or from a serial to a parallel port.

To prepare, select, or refresh device code page

MODE *device* codepage prepare=((*nnn*[*nnn*])[*path*])
MODE *device* codepage select=*nnn*
MODE *device* codepage refresh
MODE *device* codepage [/status], where

nnn is the code page number. Currently supported code pages are:

437 (United States, the default)
850 (Multilingual)
860 (Portuguese)
863 (Canadian French)
865 (Nordic)

device names are CON, LPT1, LPT2, and LPT3. The following abbreviations are also supported:

cp for codepage
prep for prepare
sel for select
ref for refresh
/sta for /status

There are five code page information (.CPI) files included with MS-DOS, which provide support specific to the following devices:

File	Device
EGA.CPI	Enhanced graphic adapter (EGA) or IBM PS/2
LCD.CPI	IBM PC convertible liquid crystal display
4201.CPI	IBM ProPrinter and ProPrinter XL
4208.CPI	IBM ProPrinter X24 and ProPrinter XL24
5202.CPI	IBM QuietWriter III printer

This is just one aspect of MS-DOS's multilingual support. For more information, see the NLSFUNC and CHCP commands.

To set display mode

MODE *display,n*
MODE [*display*],*shift* [,*test*]
MODE CON[:] [cols=*m*] [lines=*n*], where:

display is one of the following:

40 or 80 indicating number of characters per line.

BW or CO for color graphics adapter with color disabled (BW) or enabled (CO).

MONO specifies monochrome display adapter with standard 80 characters-per-line width.

n (or) line=*n* specifies the number of lines of text displayed. Valid values, usable only if supported by the video adapter, are 25, 43, or 50 lines. It also requires that ANSI.SYS or similar ANSI driver) be loaded.

m (or) cols=*m* specifies characters per line. Supported values are 40 and 80 characters only.

shift specifies whether to shift a color graphics adapter display horizontally. L (for left) or R (for right) are valid.

test, where T is the only supported value.

Usable only with CGA displays, and therefore of little use in today's environment, this kind of MODE support is still offered for those who need it in 5.0.

The *shift* option is rather poorly documented. It displaces the entire display by 1/40 of the screen width as an aid to centering the display when needed. If the problem is severe, the command can be repeated if movement to the right is required, but it can be issued only once to the left.

The *test* function is not documented at all except for the fact that it exists. It simplifies the task by displaying a test pattern. The only supported value for *test* seems to be T. The syntax is MODE ,L,T, which clears the screen and returns a display that consists of a line of numbers, 0 through 9, repeated the entire width of the screen, and the following prompt:

```
Do you see the rightmost 9? (Y/N)
```

Again, however, movement to the left is limited to 1/40 the width of the screen, and any attempt to shift it farther to the left returns this prompt:

```
Unable to shift screen left
```

To set keyboard typematic rate

MODE CON[:] [rate = *r*] [delay = *d*], where:

r specifies the rate in clock ticks. Valid entries are 1 to 32, corresponding roughly to a like number of characters per second.

d specifies how long you must hold a key down before DOS begins repeating at the specified rate. Values are 1 through 4, representing 1/4 second intervals.

Another of MS-DOS's more obscure and little used features, using MODE to set the keyboard typematic rate generates results that vary widely from documented rates and expectations.

MORE

External (.COM)

Function Limits screen output to 23 lines at one time (the default).

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format MORE <*source*
or *command* | MORE

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Supported.

MORE should probably be spelled LIST, because its most common usage is in lieu of a LIST command—something MS-DOS has never had. DOS should be smart enough to know mere mortals can't read 23 lines of scrolling text or so per second. But it isn't, and you have to force it to pause every 23 lines—or 43 or 50 lines (provided ANSI.SYS is loaded and the display device and adapter support higher density screen modes)—making the computer wait until you're ready to move on.

As you can see, MORE can be used either as a stand-alone command, typically instead of TYPE (or optionally with a file as its source), or in combination with another command. To display a file, a document for instance, on screen, you have two choices with MORE. Either of these command formats achieves the same end—displaying one screenful and then freezing the display until you type a key, any key, to move on to the next screenful, like this:

TYPE *docfilename.ext* | MORE
or MORE < *docfilename.ext*

When using the pipe character (|) in conjunction with MORE, it can operate on commands such as DIR, SORT, or TYPE. However, only files can serve as valid sources for redirection (<).

Note that, to operate, MORE has to create a temporary file on the default directory. Using a freshly formatted floppy as a default disk, three of them are shown in Fig. 20-18. As you can see, two got away—MS-DOS normally closes them out when finished with them, leaving only cryptic filenames cluttering your disk. But I caught one red-handed, created when I used MORE (instead of /P) to keep a long directory listing from scrolling too fast to read (see Fig. 20-19).

This is just an excerpt from the file, which includes a place where MORE had

```
Directory of B:\
APAJBLAE.            0 .A..  11-21-90   2:09:26 pm
APAJBLGC.            0 .A..  11-21-90   2:09:28 pm
ARDDBJCC.         1638 .A..  11-21-90   2:57:47 pm
```

20-18 MORE has a nasty habit of cluttering your hard disk with empty files that have cryptic names. Here I caught one "red handed" before MORE had a chance to flush the evidence.

```
DV          BAT          55 10-11-90    9:27a
INST        BAT         103 10-05-90    3:49p
KBD01       SYS       15480 09-21-89    5:37p
MASTER                  107 09-25-90   12:11a
Press any key to continue . . .

(continuing C:\)
MIRROR      FIL       50688 11-21-90    8:28a
MIRROR      BAK       50688 11-20-90   10:33a
OLD         C         41472 03-19-90    9:15p
```

20-19 Excerpt of contents of one of MORE's elusive, cryptically-named, temporary files—ARDDBJCC in Fig. 20-18—clearly reveals the source of this annoyance.

inserted a screen-break point in the file. Those familiar with MORE from pre-5.0 releases will probably notice the new *Press any key to continue...* prompt, which replaces the more ambiguous —*More*— from earlier versions.

Technically, MORE is classified as a filter; for more about MORE, see the FILTERS discussion elsewhere in this book.

MSDOS.SYS

Hidden system file

Function As the name implies, this is the very essence of DOS.

Usage Can be loaded only by the system during bootup.

Origin Version 1.0. Only the PC-DOS equivalent actually dates to version 1.0. MS-DOS didn't emerge until later, when the clone market and the need for a generic rather than device-specific DOS began to develop.

Format N/A.

MSDOS.SYS in MS-DOS is functionally similar to IBMDOS.COM in PC-DOS. At the system level, the .SYS and .COM extensions can be used interchangeably. Together, with its companion IO.SYS, these two carefully protected files, invisible to the casual observer even in ordinary directory listings, are the very heart and soul of DOS. Without them, you don't have DOS and without both intact and loaded properly, a machine won't finish booting as a DOS machine.

These are the only two files that can't be replaced by functional equivalents from 3rd-party sources without having the machine, in fact, run under some operating system other than MS-DOS.

Because MSDOS.SYS and IO.SYS (or the PC-DOS equivalents) are so crucial to system operation, the R, S, and H attributes are all set during installation to prevent inadvertent alteration or erasure. Also, because they're loaded before the system is fully operational and in fact are required to bring it to a fully operational level, they must reside at specific locations on any disk for that disk to be bootable.

NLSFUNC

External (.EXE)

Function Provides support for Code Page Switching using the internal CHCP command. It also specifies country information.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.3 and later.

Format NLSFUNC [[*d:*][*path*][*filename*][*.ext*]]

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Supported.

This is another of those special utilities MS-DOS provides now to extend support for foreign language code pages.

Prior to version 3.3, Microsoft simply supplied the appropriate language support for the country where that copy of DOS was sold. In the US you automatically got the US version, but a copy bought in Paris would support French. In time, this proved to be less than a perfect answer, so the idea of switchable code pages was born. Code page switching is still not a perfect answer either, because it requires specific corresponding hardware support from display adapters and printers.

While code page support has added an important new dimension to MS-DOS, it's something few users need to be concerned with, let alone have tying up a chunk of hard disk space. For those who need the information, I discuss the several commands and MS-DOS utilities supporting code page switching in greater detail elsewhere in this book.

PATH

Internal to COMMAND.COM

Function Establishes a user-defined path for command searches.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format PATH [*d:*][*pathname*][;*d:*][*pathname*]...
or SET PATH = (to cancel the path)

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

Network use Supported.

One of the most useful of all MS-DOS commands, PATH is also surely one of the most misunderstood and most abused.

After checking commands against COMMAND.COM's internal list, DOS looks next to the default directory, regardless of whether that directory is specified in the defined PATH or not. Only if no match is found to that point does DOS look to the PATH.

Simply put, the PATH defines a search route for DOS to follow through whatever tangled web of directories and subdirectories you've created, particularly on your hard disk. The PATH can be defined or altered from the DOS command line at any time—generally, it's most conveniently defined during bootup by an entry in the AUTOEXEC.BAT. It appears, following, as just one of several entries in the environment as displayed by the DOS SET command:

```
TODAY=01-20-1991
PATH=C:\DOS;C:\DOS\BATCH;C:\DOS\UTILITY
COMSPEC=c:\dos\command.com
PROMPT=$p$_
```

The most serious drawback of PATH, as defined in this manner, is that DOS can use it to locate only files it recognizes as *executable*, like .COM, .EXE, and .BAT. This differs from APPEND, an external command, which defines a search path DOS can use to locate data and other files, as well—although there are also certain drawbacks and limitations when dealing with APPEND—see the discussion of the APPEND command earlier in this chapter.

Despite the inability of DOS to track files other than executable files over the PATH as defined in the environment, you can access any file not in the current directory by including the full (or essential parts) of the path in the call-up com-

mand. You could, for example, display a file that's in a directory other than the default directory by entering a command like this:

```
MORE <D:\TAB\DOS5\ONCEOVER.SE4
```

or, if you can read real fast:

```
TYPE D:\TAB\DOS5\ONCEOVER.SE4
```

While the PATH defined in the environment can be anything up to 127 characters in length, a good PATH is the *shortest* possible PATH you can live with comfortably. Yet, given free rein, it seems like all the applications most people install on their hard disks—and a lot of utility packages as well—not only want their own directories, but insist on adding all of those directories to the path. Right now, I've got something like 55 directories on my main working hard disk. And pretty soon, instead of simplifying command entry and speeding things up, your long, looonng, looooonnnnng path really starts to bog you down.

The problem here is that, whenever you call for a command COMMAND .COM can't find internally or on the default directory, it starts working its way laboriously through your predefined PATH, searching every directory you've listed, one by one, until it finds—or doesn't find—a match. DOS can't even come back with *Bad command or filename* until it's done the whole routine, and the more directories it has to search, the longer that takes.

You don't need all of those directories in your PATH, however—and you don't have to do a lot of extra typing either if you do it right. For instance, to speed things I keep mine stripped down pretty much to bare essentials, based on my work habits:

```
PATH=C:\DOS;C:\DOS\BATCH;C:\DOS\UTILITY
```

That's it. Out of all those 55 directories, only three are in the path. The DOS directory is first because it's probably the most used, with the one for batch files running a close second—that's because simple batch files (and DOSKEY macros) can do most computer jobs faster than you could manually. Here, for example, is a batch file called ATTACH.BAT that executes a program with the same name, which exists in a directory not in the path:

```
@C:\DOS\SQUISH\ATTACH %1 %2
```

This executes the program, passing a parameter and using the exact same syntax as if the program were in the default directory or path. And a similar memory-resident 5.0 DOSKEY macro, shown below complete with the command line that loads it into memory, does the same thing even faster. The ampersand (@) cannot be used in DOSKEY macros like it is in batch files to prevent the command finally executed from echoing on the screen.

```
DOSKEY ATTACH=C:\DOS\SQUISH\ATTACH $1 $2
```

Except when spanning multiple drives or accessing programs on other than the normal default drive, the inclusion of the drive in the path statement is optional. It's generally advisable, however. If you haven't specified the drive with every path you've specified, your path is valid only as long as the drive you assume for the path remains the default drive. Change to a different drive—and the path will be meaningless.

Note that the path, once defined, can't be edited *per se*. Entering any new PATH statement totally replaces the old path in the environment. When working with a secondary command processor—typically one loaded by an application program when you shell to DOS—any changes made to any items in the environment are made only to a copy of the master environment inherited by the secondary command processor. The master environment—the environment belonging to the command processor that's loaded when DOS boots up initially—is unaffected.

PAUSE

Internal to COMMAND.COM

Function Suspends command execution.

Usage For use only in batch files.

Origin Version 1.0.

Format PAUSE

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Not supported.

A simple function dating to DOS's rude beginnings, PAUSE has many applications. Used properly, it can automatically interrupt batch file execution if critical errors are detected—errors that in many cases might jeopardize the integrity of important files or data. Inserted temporarily between a series of batch file commands, it can also be an indispensable debugging tool.

One if the most useful applications of PAUSE is to use it in conjunction with ERRORLEVEL to interrupt batch file execution when some critical error is detected, as shown here:

```
IF ERRORLEVEL n ECHO serious error message
IF ERRORLEVEL n PAUSE
next command
```

Unless the specified error is detected, the first line doesn't echo the message and the second line doesn't interrupt the normal processing flow, skipping right on down to execute the next command. If that error is detected, however, the message is displayed and the system pauses, prompting:

```
Press any key to continue...
```

Unfortunately, this prompt is rather ambiguous because what it doesn't tell you is that you don't have to continue at that point. Allowing the process to simply continue after pausing could lead to serious consequences—in some cases even to the loss of important files or data, which would defeat the whole purpose of the interruption.

What the PAUSE prompt should tell you but doesn't is that holding down the Ctrl key while you press either the Break key or the letter C at that point, instead of just any old key, will return the message:

```
Terminate batch job (Y/N)?
```

Either of these double key entries will cause ^C to be displayed on screen. This isn't the same as entering the caret symbol and letter C, however, even though on screen it might appear like that—DOS, in this case, would interpret the caret (^) simply as "any key" and continue execution. Fortunately, most old DOS hands who know this (although many of us are "old DOS hands" only after screwing up on things like this). If anyone who isn't savvy to the ways of DOS is likely to run your batch files, it's not a bad idea to mention, in the *serious error message* you can have your batch file echo, that there's a back way out of this one—something like:

```
...ECHO Critical error detected! Urgent! Press Ctrl – C to abort.
```

PRINT

External (.EXE)

Function Prints a text file. Can run as background operation while other MS-DOS commands are processing.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.x.

Format PRINT [*d:*][*path*] *filename* (uses defaults, but prompts for the device)
or PRINT *switch(es)* [*d:*][*path*]*filename* (see text for limitations)
or PRINT [*d:*][*path*] *filename* /c
or /p (to add or remove files from queue)
or PRINT /t (to remove all files from the queue)
or PRINT (to list the current contents of the queue on screen)

Switches

/? Displays on-screen help including syntax, switches, etc.
/t Eliminates all files from the PRINT queue.

Switches in the following group precede the filename in the command:

/d*device* Specifies the print device. Supported devices are LPT1 (or PRN) through LPT3, and COM1 through COM4. PRN is the default. The /d: switch *must* precede any other switches that require a value to be supplied following the colon.

/b*size* Sets the PRINT buffer size in bytes. Allowable range is 512 (the default) to 16384. Increasing buffer size speeds printing, but reduces RAM available for other applications.

/u*value1* Sets maximum number of clock ticks the printer will wait for if the print device is not immediately available. Allowable range is 1 to 255, with a default of 1.

/m*value2* Sets maximum number of clock ticks PRINT can take to print a character before issuing error message. Allowable range is 1 to 255, with a default of 2.

/s*timeslice* Sets the number of clock ticks the MS-DOS scheduler allots to PRINT. Allowable range is 1 to 255, with default of 8. Setting a higher value speeds printing but slows processing of other concurrent applications—see MULTITASKING.

/q:*qsize* Sets the maximum number of files the PRINT queue can accommodate, to a maximum of 32. Default is 10 with minimum of 4.

Switches listed below must follow a filename when issuing command:

/c Removes the filename preceding it from the PRINT queue.

/p Adds the preceding filename as well as any that follow it to the PRINT queue.

Errorlevel Not supported.

Network use Supported.

On-Screen Help Yes.

A TSR dating to DOS's early days before the name *TSR* came into vogue, PRINT can be a very useful utility, but needs a little understanding. It's the way most README and miscellaneous documentation files are meant to be printed if you want a hardcopy. It's generally *not* the way to print word processed files, however. PRINT works a lot like the DOS TYPE command in that the way a file displays on screen with TYPE is the way PRINT will display the file on paper.

If, for example, you want to print a plain unformatted ASCII file, that's what you'll get on paper—an unformatted printout. Word processor files often have special control characters mixed in with the text—things you'd never see printed by your word processor and don't want to see in print—but PRINT doesn't know the difference and you get it all. For example, the first two lines of text from this command section came out like this with the PRINT command:

```
A TSR dating to DOS's early days before the name TSR came into vogue
```

Here, PRINT got just the first 80 characters it found—including a dozen XYWRITE control characters (not the actual printer codes normally sent to the printer, but rather codes sent to the printer driver software for translation into the proper codes for that particular printer) that make the name TSR italic. No margins, nothing. And without a carriage return in the file, the rest just went off into limbo. Yet, just the day before I had used PRINT on the same printer to print out nearly 200 pages of preformatted software documentation and it had done the job perfectly.

A quick way to see if what you want is formatted suitably for PRINT is to preview the file with the TYPE command, preferably piping it to MORE—or better yet just inputting the file as the source for MORE, like this:

```
MORE <filename[.ext]
```

You could also use PRINT itself to preview a file and see how it's formatted by specifying CON as the output device. It's painfully slow, however, and, worse, tends to ignore all efforts to break out of it (Ctrl−C or Ctrl−Break) until it has laboriously printed the entire document to the screen, leaving you little choice but to either take a coffee break or just throw up your hands and reboot. TYPE and MORE at least allow you to break out once you've seen what you want to see.

As you can see, PRINT has a lot of switches and you can pile them on like toppings on a pizza. Individual print queue entries can contain up to 64 charac-

ters. Generally, I find for most purposes PRINT *filename.ext* is all that's required to do the job.

Unfortunately, if you need to customize the PRINT command, you'll find it doesn't lend itself easily to the creation of batch files or DOSKEY macros because the location of filename varies within the overall command structure, depending upon which switches are used.

Also note that you can use only the /d, /b, /u, /m, and /q switches the first time the PRINT command is issued after booting your system. Reissuing any of these will return an error message. PRINT, remember, is a TSR and these switches affect the way it's configured. DOS provides no way to remove PRINT from memory and restart it—other than rebooting—once installed. There are various 3rd-party TSR managers that do allow for the removal of TSRs from memory without rebooting, but they aren't supported by Microsoft.

ERRORLEVEL is also not supported. While this would seem especially useful with PRINT, keep in mind that PRINT supports nearly 20 internally-generated error messages, which seems sufficient explanation.

PRINTER.SYS

External (device driver)

Function Supports code page switching for parallel ports.

Usage Can be loaded only from the CONFIG.SYS file.

Origin Version 3.3

Format DEVICE = [*d:*][*path*]PRINTER.SYS lptx = (*type*[*code*[,...],*n*]) where *x* (from lptx) specifies the number of the port for which you want code page switching supported, and *type* specifies the printer type.

Printers currently supported are:

4201 (IBM 4201 ProPrinter family)
4201 (IBM 4202 ProPrinter XL)
4208 (IBM 4207 or 4208)
4208 (IBM ProPrinter X24 or XL24)
5202 (IBM QuietWriter III)

the variable *code* specifies the code page(s) supported, currently including:

437 (United States)
850 (Multilingual)
860 (Portugal)
863 (Canadian-French)
865 (Norway)

and *n* specifies the number of additional code pages the hardware will support.

Switches None.

Errorlevel Not supported.

Network use Not supported.

Another set piece in DOS's multilingual puzzle, PRINTER.SYS is one of the more straightforward of the several device drivers and other programs specific to multilingual usage.

PROMPT

Internal to COMMAND.COM

Function Allows you to customize the MS-DOS command line prompt.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format PROMPT *option(s) and/or text*
or PROMPT (to restore the default, C>, prompt)

Switches

/? Displays on-screen help including syntax, character codes, etc. The character codes are as follows:

$q equal sign
$$ dollar sign
$t current (system internal clock) time—reads time at entry but doesn't update
$d current (system internal calendar) date
$p current drive and path
$v MS-DOS version
$n current drive only
$g greater-than sign
$l less-than sign
$b bar or pipe character
$_ linefeed
$e ASCII 1Bh (the Escape character)
$h backspace to erase a character written to prompt line (displays cursor only)

Errorlevel Not supported.

Network use Supported.

A handy feature, you can customize your command line PROMPT in an almost infinite variety of ways—and if the options listed above aren't enough to suit you, PROMPT accepts almost any character string you care to throw at it. How about this one, guaranteed to lose any friends you still have at the office:

```
prompt THIS IS JIM'S COMPUTER—KEEP YOUR GRUBBY HANDS OFF!!!
```

And if ANSI.SYS is loaded, you can spell it out in livid color, reverse the video mode, and have all kinds of fun with it. Anything other than DOS's default prompt

is stored as a variable in the environment, as shown here, displayed using the SET command:

```
COMSPEC=C:\COMMAND.COM
TODAY=01-23-1991
PATH=C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY
PROMPT=$P$_
```

It's safe and easy to experiment with, too. The prompt can be changed at any time and changed again as often as you like during the course of a single work session. Once you have a PROMPT you like, the easiest thing is just to include it in your AUTOEXEC.BAT so it's installed automatically whenever you boot your system—you still have the option of overriding it any time you like.

Because PROMPT accepts ordinary ASCII input as well as its own special codes, you can combine the two in any way you like—three actually, if you include the additional ANSI.SYS options (see ANSI.SYS). If you butt options together without spaces separating them, that's the way they'll appear. You can also add spaces, or with the $_ you can display parts of the prompt on different lines.

I like to keep mine simple, leaving the screen as uncluttered as possible. But I also like to know what the current default drive and directory are, so I use the $p. I also happen to like to drop the command line down one line from the prompt, so I add $_ which gives me a prompt that looks something like this:

```
D:\TAB\DOS5\
—
```

Nice and simple. But it's your screen, so have at it.

QBASIC

External (.EXE)

NEW!

Function An interpreter that reads instructions and programs written in BASIC, and interprets them as executable code.

Usage Can be loaded or run from the DOS command line or batch files.

Origin Version 5.0.

Format QBASIC [*switch(es)*][/run [*sourcefile*]]

Switches

/b Allows use of a monochrome monitor with a color graphics card.
/editor Invokes the (also new) MS-DOS editor.
/g Provides fastest update of a CGA monitor.
/h Displays the maximum number of lines possible for your hardware.
/mbf Converts the intrinsic functions mks$, mkd$, cvs, and cvd to mksmbf$, mkdmbf$, cvsmbf, and cvdmbf, respectively.
/nohi Allows use of a monitor without high intensity support.
/run Runs a program file in QBASIC before displaying it.
sourcefile The program file to load or run.

Errorlevel Not supported.

Network use Not supported.

Note that there is extensive on-screen help available from within the program. A far cry from the ROM-based "cassette BASIC" of the original PC-DOS and a significant advance from the GWBASIC found in various earlier MS-DOS releases, this program stops just short of being great. Loosely translated, read that as stopping just short of having all the features found in Microsoft QuickBASIC, which it strongly resembles. It doesn't, for example, compile code written in BASIC into .EXE files, as does its big brother.

Still, QBASIC provides a great set of training wheels for budding programmers, as well as a solid foundation for EDIT, the new MS-DOS editor, also introduced with this release. QBASIC is too big and complex to treat with in any detail here, but I've discussed it in greater detail elsewhere in the book.

If you're an old BASIC user and have BASIC programs written for GW BASIC or BASICA, they can, in most cases, be easily updated to run under the new QBASIC interpreter.

RAMDRIVE.SYS

External (device driver)

Function Creates a virtual disk in RAM for fast volatile data storage.

Usage Can be loaded only from the CONFIG.SYS.

Origin Version 3.2.

Format DEVICE = [*d:*][*path*]RAMDRIVE.SYS[*disksize*][*sectorsize*][*entries*] [*switch(es)*])1

where *disksize* specifies the size in Kilobytes (16-4096), the default being 64; *sectorsize* is the size of sectors, with valid sector sizes being 128, 256, and 512, and the default 512; and *entries* is the number of files (and subdirectories) you can create in the root directory of a virtual disk. The allowable range is 2-1024; the default is 64.

Switches

/e Creates RAM disk in extended memory rather than expanded or conventional memory.

/a Creates RAM disk in expanded memory rather than extended or conventional memory.

Errorlevel Not supported.

Network use Not supported.

Despite the complex looking syntax, RAMDRIVE.SYS is really easy to install. Unless you want a tiny 64K RAMDRIVE, however, you do have to specify the *disksize*. And it's unlikely you'd want to use any of your precious conventional memory, so you'll probably want to use one of the two switches. But go with the defaults for *sectorsize* and *entries*. In the real world, then, a typical CONFIG.SYS line would read something like this:

```
DEVICE = \DOS\RAMDRIVE.SYS 1024 /e
```

With this, you've got a 1-megabyte (1024K) RAM disk created in extended memory, loading RAMDRIVE.SYS from the DOS directory. The size can be scaled down to fit a system with less memory to spare, but even if you can afford only the default 64K, you'll find a RAMDRIVE can be surprisingly useful if you're not already using one.

The good news and the bad news about RAM disks in general is the fact that they're volatile—that is, if the power hiccups, your RAM disk and everything on it is history. The bad side is that you don't want to use a RAM disk for anything that's very important. But the good side is that, when you're finished working,

any junk you put on your RAM drive simply goes away without cluttering your hard disk.

I use a RAM disk extensively as the target for redirected output from sorts and searches—data I need only during the current work session and probably never again. And one of the handiest uses for a RAM disk is as the default disk for the temporary files—typically the ones with the .TMP extensions—a lot of programs like to clutter up your hard disk with.

Speed, of course, is certainly one of the biggest advantages of RAM disks over other storage media. They're much faster than the fastest hard disks and floppies, which hardly get their act together by the time a RAM disk finishes, as shown by the following comparison of the times required to copy a roughly 80K file to different media:

To RAMDRIVE	To hard drive	To 720K floppy
01.37 sec.	03.40 sec.	19.11 sec.

Assuming a large enough RAM disk, this speed advantage can be put to good use with programs that require extensive I/O by copying the whole program to your RAM disk and running it from there. A copy of COMMAND.COM on a RAM disk with the COMSPEC set to read that copy can speed many operations significantly. And the list goes on and on. If you're not using RAMDRIVE yet, don't you think it's time?

RECOVER

External (.EXE)

Function Recovers data from bad or defective disks.

Usage Should be executed only from the DOS command line.

Origin Version 2.x.

Format RECOVER [*:*][*path*][*filename*[*.ext*]]
or RECOVER *drive* (DANGER! Use in this way only with caution)

Switches

/? Displays minimal on-screen help, but go to the documentation for this one.

Errorlevel Not supported.

Network use Not supported.

Probably one of MS-DOS's most deadly dangerous commands, RECOVER attempts to read files on a sector-by sector basis, recovering whatever data is salvageable from good sectors. Data from bad sectors is, of course, lost.

Often—and with any luck—you'll be able to RECOVER most if not all of your actual data. However, especially in the case of spreadsheets, databases, and other applications that use special file structures. Recovery of raw data might not by itself assure that any of the data will be usable. Figure 20-20, for example, is an excerpt from the check register of an electronic bookkeeping program I use. Even with the entire file intact, I couldn't find the amounts of the checks, the dates, or very much else about these items in legible form anywhere. The moral to the story, then, is use RECOVER only as a last resort—and even then use it with care.

```
$181  VanNote Lumber Co
                Office cabinet materials          $
v%182  Barnes & Noble
                Reference book order              %
&183  Time Life books
                "UFO's"                           &
h(184  Maryland Bank NA
                AOPA Master/Visa                  (
*185  F-Stop Shop
                Film, Camera supplies             *
```

20-20 Even if RECOVER is able to recover a complete file, the data might be unrecoverable, as in this excerpt from a checkbook register.

There are two ways to use RECOVER. The most benign and generally most useful way is to recover whatever is recoverable from one or more *specific* data files that suddenly won't load. Of course, before even doing that you should back up and make sure it's not the program that's become corrupted and not the data files themselves.

Of course, if you do recover a file that was good in the first place, you're probably not in any trouble. Run on a single file, it doesn't even change the filename and, if all the data was intact before you ran RECOVER, it will—or should—still be intact when you're finished. Still, as with any operation that can alter files or data, you should try to make a copy of any files you think are damaged *before* you try to fix them . . . just in case.

I wouldn't even try to recover programs. If a program is corrupted, it's gone. If it wouldn't run before you ran RECOVER, you can bet whatever you can salvage won't run either. Your best—realistically your only bet—is to go back to the distribution disk, your original source, and get a fresh copy to work with. If it's one that comes as part of a compressed package, you might want to reinstall it in a different directory, particularly if there are any configuration files associated with it—and with a little luck you can salvage and reuse these.

Where RECOVER is most dangerous—and most likely to get you into trouble—is if you try to use it on an entire disk rather than a single file (or simply forget to specify a filename after telling it what drive you want it to attack). It will

```
 Volume in drive B is unlabeled
 Directory of  b:\*.*
file0001.rec     3072  12-03-90  15:37
file0002.rec     1024  12-03-90  15:37
file0003.rec    27648  12-03-90  15:37
file0004.rec    53248  12-03-90  15:37
file0005.rec    35840  12-03-90  15:37
file0006.rec    59392  12-03-90  15:37
file0007.rec     7168  12-03-90  15:37
file0008.rec    31744  12-03-90  15:37
file0009.rec    17408  12-03-90  15:37
file0010.rec     3072  12-03-90  15:37
file0011.rec    44032  12-03-90  15:37
file0012.rec     1024  12-03-90  15:37
file0013.rec     8192  12-03-90  15:37
file0014.rec    53248  12-03-90  15:37
file0015.rec     8192  12-03-90  15:37
      354,304 bytes in 15 file(s)
      375,808 bytes free
```

20-21 All that remains of the root directory of a disk after careless use of the RECOVER command. Filenames and all trace of subdirectories are gone.

```
 Volume in drive B is unlabeled
 Directory of  b:\*.*
drvtype.exe       2894   11-10-90   13:40
hinstall.bat       103   11-01-90   13:00
install.exe      26849   10-26-90   16:29
install.sop      53234   11-23-90   10:37
readme.doc        7573   11-23-90   13:40
scheck.exe       34872   11-21-90   17:54
screxec.exe      59199   11-23-90   10:54
sdiag.exe         6848   10-22-90   15:35
sdir.exe         31042   11-21-90   17:53
sremove.exe      16646   11-21-90   18:31
stacker.lbl         92   11-23-90    7:32
swapmap.exe       7423   11-21-90   17:29
tpov.exe         53221   10-24-90    8:12
xswap.com         2554   11-20-90   14:27
xtacker.com      43246   11-21-90   14:41
      354,304 bytes in 15 file(s)
      375,808 bytes free
```

20-22 The same disk in Fig. 20-21, but after UNFORMAT is run, which fully restores its condition prior to careless use of RECOVER.

turn the entire root directory into instant mush, as shown in the directory of a floppy disk in Fig. 20-21.

No filenames and, except for size, you can't tell files from subdirectory entries—and even then you can't be sure. Until DOS 5.0, this might have been the end of the story—would have been if some of those really were subdirectory entries. Fortunately, however, DOS 5's UNFORMAT can unmush it, so it's not the one-way street it used to be. Figure 20-22 shows the same mushed root directory rebuilt.

REM

Internal to IO.SYS

Function Allows the entry of comments in batch files.

Usage Can be in batch files and is now supported in CONFIG.SYS, as well.

Origin Version 1.0.

Format REM *comment*

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Supported.

DOS 5.0 for the first time supports the use of REM (with us from the beginning of DOS) to insert comments or other nonexecutable lines in the CONFIG.SYS, as well as ordinary batch files. Although not officially supported or documented, REM can be used in the CONFIG.SYS with version 4.x files, also.

Many users think of REM only in terms of adding comments to their batch files, as in this excerpt from my AUTOEXEC.BAT:

```
REM if !%TODAY%= control passes to \dos\batch\current.bat, runs
optune, date >
```

That's there to remind me where to look in case other functions the AUTOEXEC is supposed to trigger by using CALL statements don't work. And you'll note I just sort of ran off the page with it. This REM actually wraps requiring two lines to display it which is fine as long as there is no carriage return to make DOS think the second line is in fact a separate line.

Or you can use REM at the beginning of successive lines if you prefer. Either way is okay. The only thing that's crucial when using REM is leaving at least one blank space after it to separate it from whatever else is on that line. If I take the space out of the line shown above, for example, DOS will return:

```
Unknown command "remif"
```

There's another use for REM that's probably more useful than just throwing in the odd reminder, however. What REM really does is tell DOS whatever follows on that line is none of its business and to skip on by. So it's a great way of taking things out of batch files temporarily—things you think you might want to put back later on. You hear people talking about "REMing" something out sometimes; this is what they mean.

Along the same line, I have to make a lot of changes in my configuration test-

ing new software and hardware, as in the following:

```
DEVICE = C: \ DOS \ QEMM \ QEMM386.SYS RAM ROM EXCLUDE =
C800 – CBFF AU DMA = 32
REM DEVICE = C: \ DOS \ QEMM \ QEMM386.SYS RAM ROM AU DMA = 32
```

REM, officially supported for the first time by 5.0, is a big help in doing this. Often it's just a matter of changing parameters—I have to try some new ones, but I want to save the old ones to go back to. REM makes it easy.

RENAME (or REN)

Internal to COMMAND.COM

Function To rename a file or files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.1.

Format RENAME [*d:*][*path*]old_name[.*ext*]new_name[.*ext*]
or REN [*d:*][*path*]old_name[.*ext*]new_name[.*ext*]

Switches

/? Displays limited on-screen help.

Errorlevel Not supported.

Network use Not supported.

An almost remarkably simple command—simple and forgiving—RENAME can be used on a single file or all files in a directory matching wildcard criteria.

You'll notice the drive and path are not included in the FORMAT shown above, and that isn't an omission. Once you've told DOS where to find the file, that's all it needs. And, unlike some 3rd-party utilities, RENAME can't pop files into any other directory instead of where you had them, even if you wanted it to. It can only change their names. That's all. However, it's forgiving in that if you should specify the drive and path as well as new filenames, DOS just ignores it—provided you specify the drive and path where it resides. Try to change it and you'll get something like this:

```
REN E:TEST.BAT C:TEST
NOT SAME DEVICE "E:\TEST.BAT"
```

RENAME won't change the names of subdirectories, either. DOS doesn't provide any means of doing that except by simply creating a new subdirectory, copying files out of the old one into it, then deleting the files out of the old directory—after making sure they were copied properly—and finally deleting (RMDIR or RD) the old directory itself.

Also, RENAME can't be used to rename a disk, but DOS does provide a means of doing that with LABEL.

REPLACE

External (.EXE)

Function Replaces files on target disk or adds files selectively.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.2 and later.

Format REPLACE [*d:*][*path*]*filename*[*.ext*] [*d:*][*path*][*switch(es)*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/a Adds files not existing on target directory, and can't be used in combination with either the /s or /u switches.

/p Echoes filenames and prompts for confirmation before each file is replaced.

/r Replaces files that have read-only attribute set.

/s Includes all subdirectories of specified disk or directory.

/w Instructs REPLACE to wait until disk is inserted.

/u Updates only—does not overwrite newer files.

Errorlevel

1 Files were successfully replaced.
2 No source files were found.
3 Invalid source or target path.
5 Attempt made to access read-only file without /r switch.
8 Insufficient memory.
11 Invalid parameter or number of parameters on command line.
15 Invalid drive specification.

Network use Supported

One of DOS's newer utilities, it's also one of its very best, made even better with the addition of the /u (update) switch beginning with version 4.x. More powerful and versatile than its name implies, REPLACE can play an important role in developing an effective backup strategy.

REPLACE can actually wear two hats. In its primary role, it does exactly what it says it does—it replaces existing files on one disk (or in one directory) with files from another. Using the /s switch, it has the capability of acting on all subdirectories under the one specified. As originally implemented, there was an element of risk involved, however. Lacking time and date switches (as implemented in XCOPY) REPLACE could arbitrarily copy old files over newer ones if newer ones were on the target disk or directory. Adding the /u switch solved that in the previous (4.x) release.

Wearing its other hat, REPLACE serves an entirely different but important purpose, *adding* to the target any files existing on the source but not the target. Unfortunately, this switch (/a) can't be used on the same line as the update switch. It can be used as part of a second REPLACE command, however—another line in a batch file, as shown below for example—to make REPLACE a singularly powerful tool for keeping fully readable file-by-file backups up-to-date.

In the following, if a nonzero ERRORLEVEL is returned by either, REPLACE processing stops at that point and branches to a subroutine that analyzes the error:

```
REPLACE V:*.* b: /u
IF NOT ERRORLEVEL 0 GOTO ERROR
ECHO Old files updated successfully
REPLACE V:*.* b: /a
IF NOT ERRORLEVEL 0 GOTO ERROR
....
....
:ERROR ......
```

The fact that REPLACE supports ERRORLEVEL detection makes it especially well suited for use in batch files to automate such tasks as this—and easier to work with for updating than XCOPY, which requires the entry of a specific cut-off date.

One of the few things REPLACE can't do is write over hidden, system, and read-only files. If it should be necessary to overwrite any such files, it's necessary to first run ATTRIB before attempting to REPLACE them.

RESTORE

External (.EXE)

Function Retrieves files and directories from BACKUP files created by the DOS BACKUP utility.

Usage Should be executed only from the DOS command line.

Origin Version 2.0 and later.

Format RESTORE *d:* [*d:*][*path*][*filename*[*.ext*][*switch*]

Switches

/? Displays on-screen help including syntax, switches, etc.
/p Prevents overwriting file modified since backup.
/s Includes and restores all subdirectories on specified disk/directory.
/a:*mm-dd-yy* Restores files modified on or after specified date.
/b:*mm-dd-yy* Restores files modified on or before specified date.
/e:*time* Restores files modified on or before specified time.
/l:*time* Restores files modified on or later than specified time.
/m Restores files deleted or modified since last backup.
/n Restores files that have been deleted.

Errorlevel

0 Process completed, terminated normally.
1 Backup disk didn't contain any files matching the files specified.
2 Some files not restored due to sharing conflicts.
3 Execution terminated by user (Esc or Ctrl−Break).
4 Execution terminated due to execution error.

Network use Supported

The other half of the DOS BACKUP team, this one can be used *only* to retrieve files that have been backed up using the DOS BACKUP utility. Unlike BACKUP, however, RESTORE doesn't lend itself as easily to batch file automation. This isn't to say the restoration process can't be facilitated from within a batch file. The main advantage would be to use the ERRORLEVEL detection to report the nature of any error encountered during the restoration attempt.

With the exception of /p to keep you from inadvertently overwriting any files that might have been modified after the backup was made, switches should be passed only as dictated by the specific level of restoration required.

In the following:

```
RESTORE %1 %2 /p%3
IF ERRORLEVEL 0 ECHO Restoration of matching files successful
```

```
IF ERRORLEVEL 1 ECHO No matching files found to restore
IF ERRORLEVEL 4 ECHO Restoration terminated due to error
```

the source (%1) and target (%2) are specified using the same syntax, with any switches passed by a third variable (%3). While one other ERRORLEVEL value is supported to reflect a user abort, I didn't include it because I think most people would know it if they aborted with a Ctrl−C or Ctrl−Break without having DOS remind them.

The proper and selective use of switches is probably the most important consideration when using RESTORE. Unless you've had a total wipeout, you should never just restore blindly. "If it ain't broke, don't fix it" would seem to be the proper philosophy here because, as with any other file juggling operation, you could wind up writing bad over good. I have, in fact, on at least one occasion found a BACKUP file to contain one or more bad files while the originals were intact—a rare occurrence, admittedly, but it can and *does* happen.

If the backed-up disk has become corrupted and it's necessary to completely restore, if the disk was bootable and needs to be bootable again, DOS will have to be reinstalled separately because BACKUP can't save nor replace the IO.SYS or MSDOS.SYS (or IBMBIO.COM or IBMDOS.COM with PC-DOS) files.

RMDIR (or RD)

Internal to COMMAND.COM

Function Removes an existing directory.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.x.

Format RMDIR [*d:*][*path*]*directory*
or RD [*d:*][*path*]*directory*

Switches

/? Displays minimal on-screen help.

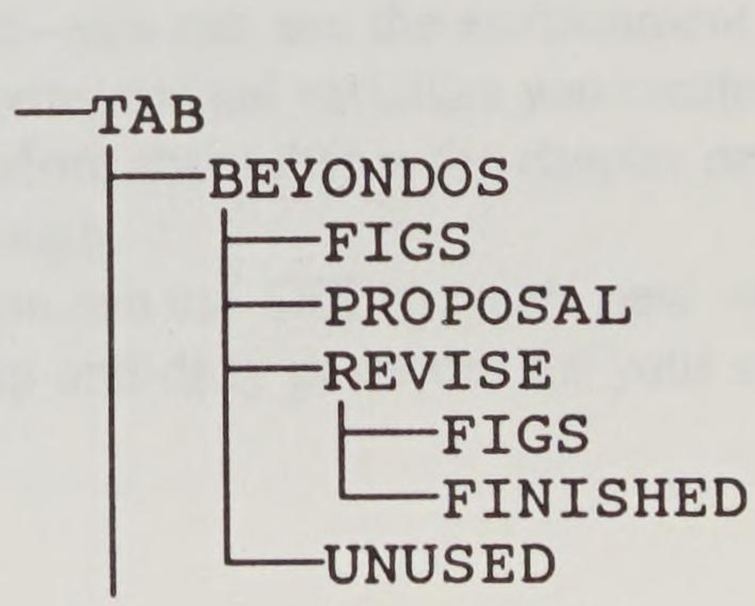
Errorlevel Not supported.

Network use Supported

While the function of this one should be fairly obvious from the name, the mechanics of using it aren't. Yes, it will remove unwanted directories—unwanted branches from your tree—but only if certain conditions are met: one, if the directory is first completely empty, two, if it's at the lowest level for that part of the tree and, three, if there are no SUBST, JOIN, or APPENDs pointing to it.

You can't remove any directory that contains files of any kind—or that DOS thinks contains files. I bring that up because DOS is especially careful about what directories can be removed and sometimes, even if there are actually no files left, DOS isn't impressed—I've got a couple of those on my hard disk now. Sometimes they contain hidden, system, or read-only files that can be removed only by changing their attribute bytes via the ATTRIB command—with DOS 5.0 for the first time you can spot these using DIR with the new /ah, /as, and /ar switches.

For those times when there's truly nothing there but DOS won't allow you to remove the directory, don't fight it. There are ways to eliminate such garbage, but none that are really recommended as they can take some serious digging and it generally isn't worth the effort.



20-23 One branch of directory tree, showing how successive levels can be used to delineate different kinds of files.

The other condition is that, rephrased, there are no lower subdirectories under the one you want to remove. Figure 20-23, for instance, is a tree excerpt from one of my hard disks, starting with TAB in the root directory. REVISE is at the same level as PROPOSAL or UNUSED but, as you can see, is not removable because it has another level under it.

The command to remove an unwanted directory must include the entire path to that directory. The command to remove the PROPOSAL directory from above the command would read:

```
RD F:\TAB\BEYONDOS\PROPOSAL
```

SET

Internal to COMMAND.COM

Function Used to display or alter the MS-DOS environment.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format SET (displays current environment)
or SET *variable*[=]*string*

Switches

/? Displays on-screen help.

Errorlevel Not supported.

Network use Supported.

The DOS environment is a little like the weather—a lot more people talk about it than have any understanding of it. It would be better likened to a scratch pad, however, because it's where DOS keeps track of things that change from day to day or even in midsession.

For example, every time you boot your system, DOS notes where in the environment it found COMMAND.COM (COMSPEC=) so it always knows where it can find it. Then there's whatever PATH you want to set and, typically, a few more things including, possibly, some that DOS didn't anticipate, as shown here:

```
SET
COMSPEC=C:\COMMAND.COM
TODAY=02-05-1991
PATH=C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY
PROMPT=$P$_
DIRCMD=/o:n/p
```

Here I also have the new DOS 5.0 DIRCMD variable set to customize my directory listing defaults, a custom prompt, and the date set as a variable. It happens to be one I actually use, but it also makes a point—you can use the environment as your own scratch pad, too, and actually use environmental variables you create in as many ways as you can think up reasons for. More about that in the chapter dealing with batch files and DOSKEY macros, though.

Of more immediate interest here is how you can use SET to create new variables. So why don't you use it to create a cheap-and-dirty password for your system:

```
SET PASSWORD=cheap-and-dirty
```

And here it is, now part of your new environment:

```
SET
COMSPEC=C:\COMMAND.COM
TODAY=12-05-1990
PATH=C:\DOS;\C:\DOS\BATCH;C:\DOS\UTILITY
PROMPT=$P$_
DIRCMD=/o:n/p
PASSWORD=cheap-and-dirty
```

You can use variables like PASSWORD or TODAY in batch files by enclosing them with percent signs, as in this line my system runs every time I boot my machine:

```
IF !%TODAY%==!%4 GOTO END
CHKDSK
MIRROR C:
```

The exclamation points are included only to guarantee that there's always something on both sides of the equation for DOS to test against, as it can't test if it finds nothing to test against. If the date in the environment matches the date in a file that tells it when the system was last started, it skips the next few steps. If the dates don't match—and they won't the first time I boot each day—the batch file checks the condition of my hard disk, runs MIRROR, and does some other housekeeping duties that take a minute or so, but that I like to have done at least once each day.

To find out how to actually use the password to keep at least most people out, refer back to the batch file and macro chapter.

I need to take a moment here to talk a little about syntax because there are some anomalies, particularly where the path is concerned. Any of the following are valid when changing your path:

```
SET PATH=path
PATH=path
PATH path
```

To set any other variables in the environment, the variable name must be preceded by SET, although the equal sign (=) is optional in all cases. Similarly, any variable in the environment can be removed with the command SET *variable*, with nothing following (except the equal sign if you choose to use it). You can even wipe out the COMSPEC if you like, but I don't advise it because that leaves you with a system that will crash with little provocation.

SETVER

External (.EXE)

NEW!

Function Sets different version numbers for executable files that don't anticipate DOS 5.0.

Usage Can be executed from the DOS command line.

Origin Version 5.0.

Format SETVER (to view current entries in version table)
or SETVER *progname.ext n.nn* (to a add new entry to version table)

Switches

/? Displays on-screen help.
/delete or /d Deletes the entry for the specified program.
/quiet or /q Display no messages—works only with /delete.

Errorlevel

0 Successful completion.
1 Invalid command switch.
2 Invalid filename specified.
3 Insufficient memory.
4 Invalid version number format.
5 Specified entry not found in table.
6 MS-DOS system files not found.
7 Invalid drive specified.
8 Too many command line parameters.
9 Missing parameters.
10 Error reading MS-DOS system files.
11 Version table is corrupt in system files.
12 Specified MS-DOS system files don't support a version table.
13 Insufficient space in version table for new entry.
14 Error writing MS-DOS system files.

Network use Supported.

An interesting addition to DOS in this release, SETVER addresses a problem some programs have running under this new release. Most programs these days require DOS 2.0 or 2.1, and some need 3.0 or better, depending on the specific DOS services they require—services that didn't exist prior to some specific release. It's a good practice for a program to check and be sure before trying to load and possibly crashing the system.

The problem—or at least one common problem—is that the way many programs go about checking the DOS version excludes the possibility of a DOS 5.0 or higher. Rather short-sighted it would seem, but nonetheless a fact of life. What SETVER does, then, is keep a list of such programs and gives them a phony version number when they ask. Cute—and here are some already on SERVER's list, plus one I've added to show you how it works:

Program	**Version**
WIN200.BIN	3.40
WIN100.BIN	3.40
WINWORD.EXE	4.10
EXCEL.EXE	4.10
METRO.EXE	3.31
IBMCACHE.SYS	3.40
OPTUNE.EXE	4.01

And this is interesting—several of Microsoft's own programs are at the top of this list, as well as Lotus's METRO, a cache driver from IBM . . . It's an interesting rogue's gallery. And DOS 3.40? Also very interesting. . . .

I added OPTUNE, a disk optimizer I use that balked at 5.0 but ran nicely under 4.01—note no drive or path is specified, only the name of the program. SETVER accepts only the filename and extension:

```
SETVER OPTUNE.EXE 4.01
```

Because I knew from experience that this program was happy with 4.01, that was the phony version I.D. I put in SETVER's version table for it. And it works just fine, as long as it doesn't know it's running under 5.0. If it hadn't worked, I could have tried other phony versions until I found one that did—either that or I ran out of possibilities.

While SETVER is an external program, the actual version table it addresses is actually internal to MSDOS.SYS. If you're into a little innocent snooping, you'll find all the entries in the version table listed there in English, using DEBUG or any utility that converts hex to ASCII. The location is significant mainly because it points out—and explains—one of the peculiarities of the scheme. Additions or changes to the version table become effective only the next time the system is booted, which makes sense because the system files are reloaded only when you reboot.

Unfortunately, not all programs will settle for such chicanery. There are actual differences in 5.0 some programs just can't swallow. This is nothing new, however. We went through this with 3.0 and again with 4.0—which was so bad that 4.01 came out as soon as 4.0 hit the shelves.

Mercifully, overall I've found 5.0 to cause far fewer compatibility problems than some earlier releases—even at the early beta testing stage before other soft-

ware developers have had a chance to cater to any peculiarities it might have. Inherently, 5.0 seems to be a more generally compatible release. SETVER just helps smooth over some of the rough spots and some of the short sightedness in certain quarters, including some that did surprise me.

SHARE[.EXE]

External (.EXE)

Function Provides support for file sharing, locking, disk change detection, and hard disk partition larger than 32 Mb.

Usage Can be executed from the DOS command line or in batch file.

Origin Version 3.0 and later.

Format SHARE [*switch(es)*]
or INSTALL = SHARE [*switch(es)*] (to load from CONFIG.SYS)

Switches

/? Displays on-screen help including syntax, switches, etc.
/f:*nnn* Sets aside RAM for sharing codes and locks (default is 2048 bytes).
/L:*nn* Sets aside memory for locks (default is 20 locks).

Errorlevel Not supported.

Network use Supported.

On-screen help Yes.

Typically, you would use this one only in a networking environment or in certain multitasking situations—those, for example, where certain files are shared by more than one process that might be run concurrently.

Once loaded, SHARE operates invisibly. It works only with software that specifically supports use in a networking or network-like environment—and there's more involved to network versions than just a label and license fee or bigger price tag. For SHARE to play a role, programs must use DOS function 3Dh to open files and store a special sharing code in the computer's memory. This code determines what access other programs or users have to the open file. Once a program is given access, it might (through function 5Ch) lock that file (or portion) if exclusive access is required.

The only choice left you is whether you need SHARE on your system, which, if you're networking, you probably do. For most multitasking, however, you don't—except in the rare instance where files might be shared.

If you do use SHARE, you have the option of loading it either from the AUTOEXEC or other batch file, on the command line, or from the CON FIG.SYS via the INSTALL = option. This has nothing to do with the operation of SHARE. Rather, it's mainly a memory management issue and, as such, is discussed in greater detail in the chapter on memory management.

SHELL

Internal to IO.SYS

Function Specifies a command processor or parameters other than the normal DOS default.

Usage Can be used only in the CONFIG.SYS.

Origin Version 3.2.

Format SHELL = [*d:*][*path*]*filename.ext*[*switch(es)*][*string*]

Switches Note that SHELL doesn't have its own, but supports the following COMMAND.COM switches:

/c Tells the command processor to perform the command specified by the *string* and then exit (extremely rare in conjunction with SHELL).

/e:*nnnn* Specifies the size of the environment in bytes (odd values are rounded to multiples of 16). The range is 160 (default) to 32768.

/p Makes the specified copy of COMMAND.COM permanent in memory—this switch should be used only in conjunction with SHELL.

Errorlevel Not supported.

Network use Not supported.

SHELL is a term commonly applied to the command processor—also sometimes referred to as the command interpreter. DOS cannot function without having a valid command processor loaded to interpret, execute internally, route, load, and initiate actions requiring programs external to it.

COMMAND.COM is MS-DOS's default command processor and the one most commonly specified in SHELL statements. However, because COM MAND.COM is the default that the DOS kernel looks for when it loads, no SHELL statement is even needed in most cases, the most notable exception being to increase the size of the environment—which, as discussed earlier in this book, should be avoided if possible. What this says is that a lot of people have SHELL statements they don't need in their CONFIG.SYS files. They don't hurt anything, but if you don't need them . . .

The DOS kernel, however, isn't really all that fussy about command interpreters. If, for example, it doesn't find one, it reports:

```
Bad or missing command interpreter
```

Not *Bad or missing COMMAND.COM*, but rather the generic interpreter. And there are in fact several command interpreters available (from 3rd-party sources) that can be substituted without making DOS the least bit unhappy.

Without digressing into the relative merits of any such substitution, there's a

command interpreter called 4DOS, for example, that's fairly popular. To install it, simply add a CONFIG.SYS line something like this and DOS will actually march quite happily to the tune of a rather different piper:

SHELL=[*d:*][*path*]4DOS.COM [*parameters*]

This will also cause a corresponding COMSPEC= statement to entered in the environment specifying the alternate command processor there, as well.

Keep in mind, however, that the SHELL command actually loads only the *primary* command processor. This should not be confused with "shelling to DOS," a term commonly used in connection with executing DOS commands from within applications programs. That involves actually loading a secondary command processor—by default, generally just another copy of COMMAND.COM (or whatever was loaded as the primary command processor) unless some different command processor is called for by the COMSPEC.

SHIFT

Internal to COMMAND.COM

Function Changes (shifts) the position of replaceable parameters in batch file processing.

Usage Can be used only in batch files.

Origin Version 2.x.

Format SHIFT

Switches

/? Displays minimal on-screen help—you'll need the manual for this one.

Errorlevel Not supported.

Network use Not supported.

Think of SHIFT as essentially doing nothing you couldn't do by running a batch file that allowed only one replaceable parameter over and over in rapid succession, simply entering a new string—a different filename, for example—for that replaceable value every time you run it.

This is one of the lesser batch file functions. Usually, when tempted to use SHIFT, I figure that I must have overlooked a better way. Still, when all else fails, it has a valid place. As indicated above, the syntax is simply SHIFT. What it shifts is somewhere else, generally one or more lines down.

For instance, you could have a simple one-line batch file something like XCOPY %1 B: /s and run it time after time, simply passing a different filename (or wildcard group) as the replaceable parameter. Or you could do it this way:

```
XCOPY %1 B: /s/m
:LOOP
SHIFT
If !%1 = =! GOTO END
XCOPY %1 B: /s/m
GOTO LOOP
:END
```

Here you'll note the test after every shift to make sure there still is, indeed, another valid file wildcard or group to copy. Otherwise, it will just keep recopying the last file(s) until you hit Ctrl−C or Ctrl−Break. The IF test automatically breaks the loop once the last valid entry is processed.

And, again, if you've entered parameters on the command line past %9, they'll be shifted and processed anyway, so shift can do a lot for you. With a little careful organization on your hard disk, however, you can save yourself time, trou-

ble, and typing by keeping all the subdirectories of files you want to back up this way grouped under one blanket directory—the way I do with the directory FILES. Then just:

```
XCOPY FILES\*.* B: /s/m
```

One line does it all—and re-creates the whole directory structure on my backup as I go. Like I say, most times when I'm tempted to use SHIFT I find there's a better, easier way to go if I just stop to think about it for a minute.

SMARTDRV.SYS

External (device driver)

Function Creates a disk cache in extended or expanded memory.

Usage Can be installed on from the CONFIG.SYS.

Origin Version 4.x.

Format DEVICE = [*d:*][*path*]SMARTDRIVE.SYS[*init_size*][*min_size*][/a]

Switches

/a Uses expanded rather than extended (default) memory.

Errorlevel Not supported.

Network use Not supported.

Disk caching is one of those things a lot of users think they need but few understand. And beyond a general definition, it's harder to pin down than, say, a RAM disk. Roughly defined, a disk cache is a buffer that, up to the limit of its size, provides intermediate storage for data from the most recent disk reads or writes to RAM.

In the case of disk reads, there's no saving of time when a file is first read, but any re-read commands that come along while that data is still held in the cache can find and access it at RAM speed rather than laboriously rereading the data from the disk.

Writes to disk, on the other hand, benefit by being able to absorb the data at RAM speeds and passing it on to the disk as fast as the disk is able to perform the actual write. That actual write can still be going on as a background operation for some period of time after DOS thinks the write is finished and turns the system back to you. You can often see this happening—DOS (or your application) gives you back the cursor and lets you get back to work while, if you look, the drive light tells you your machine is still working.

How much a disk cache speeds things up is hard to quantify because it's hard to simulate the way different people work. In what I would characterize as a sampling of typical disk writes—typical to the way I work—the net time saved by disk caching can run as high as 30% with a 500K cache, though, I hasten to point out, it's more often somewhat less. Which brings me to the next point—it's even harder to quantify how big a cache you'll want or need.

Obviously, too small a cache will have little effect. SMARTDRV.SYS defaults to 256K, which is a respectable minimum, although more can quite often be better with disk cache programs. However, even the "more is better" approach generally seems to reach a point of diminishing returns. How much RAM you should devote to disk caching all depends on just what memory resources you have available and what your other demands are for them.

One point that's inescapable when using SMARTDRV.SYS—or any cache—is that you should reduce the number of DOS BUFFERS to no more than about 5 in most cases, despite whatever the documentation for your applications try to tell you (or worse, tries to install for you when you install their software). DOS BUFFERS are very crude caches, each the size of one disk sector. Because they're crude, they're slow—much slower than a SMARTDRV cache. And unlike full-blown disk caches, the more BUFFERS you have, the slower they are. DOS doesn't recognize this fact, however, and will always pick through the BUFFERS first, before even recognizing the fact that a cache exists: the more BUFFERS, the longer this takes before you even get to the good stuff.

Despite this, you do need some buffers even with a cache; I've found five to generally be a respectable number. But only in conjunction with a cache. If you don't use SMARTDRV.SYS or some other regular cache, you should set BUFFERS= to the maximum number your most demanding applications software calls for.

SORT

External (.EXE)

Function Sorts input data.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format SORT [*d:*][*filename*[*.ext*][*switch*][*redirect*]*source*

Switches

/? Displays on-screen help including syntax, switches, etc.
/r Sorts data in reverse order.
/+*n* Sorts data according to character in column *n* of each line.

Errorlevel Not supported.

Network use Not supported.

On-screen help Yes.

SORT is another DOS old-timer and has changed little through the years. Designed as a filter (along with FIND and MORE), it took the inherent speed and processing power of the 286 processor and higher CPU chips to make even this one respectable. It's still no speed demon, but it does have certain virtues—enough perhaps to warrant the less than 7K of hard disk space it takes up.

Sorting and resorting data is something most of us need to do from time to time, but most better applications programs today have their own sort utilities. They're not always necessarily more powerful (though many are), but are generally tailored to the needs of the specific application and can sort in ways no DOS utility can duplicate. This is especially true in the case of spreadsheets and databases, which format data in special ways, sometimes even storing it in special codes that aren't even readable except through their own proprietary software.

Obviously, if you can't read it you can't sort it, so the first requirement for using DOS's SORT command is that the data be in readable form and identifiable format.

Traditionally, sorting disk directories has been one of the most useful jobs for SORT, as well as the one most often pointed to in order to demonstrate SORT's capabilities. However, DOS 5's greatly enhanced DIR command has certainly kicked the legs out from under that one—even letting you set DIRCMD defaults in order to perform almost any kind of directory sort automatically.

There are still many applications for the SORT command, however, especially because one of SORT's strong suits is that it doesn't matter where on a line of data the particular piece you want to use as your sort criteria is located. As long as that piece is always the same number of characters from the beginning of the

```
PAGEMAKR | Aldus                        | 1-206-622-5500
CBGUIDE  | Computer Buyers Guide        | 1-212-807-8220
FAA-FSDO | FAA-FSDO,                    | 1-215-596-0673
OKIDATA  | Okidata, PR                  | 1-215-627-3535
SUPRCALC | Computer Associates, PR      | 1-408-432-1727
AUTOCAD  | Autodesk, tech support       | 1-415-331-8354
XYWRITE  | Xyquest                      | 1-508-671-0888
CANON    | Canon USA,                   | 1-516-488-6700
```

20-24 Part of a phone directory file that's been sorted by area code using the first number in the area code as the key.

line, you can sort by it. Figure 20-24, for example, is part of auto-dialer phone directory sort, in this case based on area code (beginning with the 53rd character on each line) that's generated when you issue the following command:

```
SORT /+53 <FILES\PHONE
```

Normally, you wouldn't even bother sorting so few lines, so to keep a typical sort from scrolling by too fast to be of any use you would probably want to use MORE. Or, better yet, redirect the sorted output to a file—a temporary file—so you can study it at leisure. SORT lends itself especially well to piping and redirection—in fact, that's one of the things it does best.

STACKS

Internal to IO.SYS

Function Allocates stack frames to handle hardware interrupts.

Usage Can be used in CONFIG.SYS only—generally not required.

Origin Version 3.2.

Format STACKS=*n,s* where *n* is the number of stack frames allocated. The allowable range is 8−64; the default is 0 for PC/XT-type machines and 9 for AT and PS/2 type machines. The variable *s* specifies the size of each stack frame bytes. The allowable range is 32−512 bytes; the default is 0 for PC/XT-type machines and 128 for AT and PS/2 types.

Switches None.

Errorlevel Not supported.

Network use Not supported.

This is one of DOS's more obscure commands—and another one that can be used only in the CONFIG.SYS—you can forget about until or unless you find yourself getting an Internal Stack Error message. Frankly, I think I've seen that message only once, even with all the oddball configurations I run, and that was apparently a fluke because it didn't recur.

So just be aware that stacks exist and that there's a STACKS command if your computer starts telling you you've got stack error problems.

SUBST

External (.EXE)

Function Uses a drive designator as an alias for drive and path specifications.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.1 and later.

Format SUBST *d:**path*[*path*][*path*]
or SUBST (to display list of substitutions currently in effect)

Switches

/? Displays on-screen help including syntax, etc.
/d Cancels (one) specified substitution assignment.

Errorlevel Not supported.

Network use Not supported.

This is one of DOS's best and most useful utilities. Originally implemented for use by WordStar and certain other programs that were unable to process path specifiers, SUBST can be equally beneficial in dealing with files for programs that can use path specifiers, as shown in Fig. 20-25.

```
D:
├──FILES
│     ├──SUPRCALC      SUBST S: D:\FILES\SUPRCALC\1991
│     │     ├──1989
│     │     ├──1990
│     │     └──1991
```

20-25 Example of substituting a 3rd-level directory, in this case saving 20 keystrokes.

Here I've used S: as a sort of shorthand for the name of the application. Once the drive designator has been assigned, any files in what is, in this case, a third-level subdirectory can be accessed as if they were in the root directory of drive S:. It's a lot easier to type S: than the actual path.

One of the nice things about substituted drive designators is that, unlike real ones, they don't have to be consecutive. You can skip around as much as you like, picking drive designators that suit your personal likes or needs, as shown below:

\dos\subst i: d:\tab\dos5\figures
\dos\subst j: d:\projects
\dos\subst l: d:\letters\1991
\dos\subst m: d:\magazine\guide

```
\dos\subst p: d:\personal
\dos\subst q: d:\letters\1991\queries
\dos\subst r: d:\tab\beyondos\revise
\dos\subst t: d:\projects\timeshar\book
\dos\subst u: d:\projects\travel\misc
\dos\subst v: d:\tab\dos5\finished
\dos\subst x: d:\fiction\misc\short
```

To use SUBST, you must have used the LASTDRIVE= command in the CONFIG.SYS to allow the use of drive designators beyond the five (A: through E:) DOS allows by default. I go for the maximum (LASTDRIVE=Z), not because I expect to use all 26 but because it allows me to pick drive designators that relate in some way to their purpose, so they're easy to remember.

Some purist turn up their noses at SUBST, but I don't know why. As you can see from the above excerpt from the batch file that loads my word processor, I use it because it saves me a lot of time typing (and typing errors).

Even so, SUBST should not be used indiscriminately or without understanding some of its peculiarities. For instance, commands like XCOPY and REPLACE that support the /s switch for maintaining and, if necessary, re-creating the directory structure of your hard disk when copying to floppies will treat files in the directory represented by a surrogate drive designator as if they were in the root directory of a real drive and copy them accordingly.

To maintain the integrity of the directory structure under these conditions, you have to be just a little smarter than DOS—which isn't very hard. To COPY or XCOPY the contents of the substituted V: drive to a floppy while maintaining the directory structure, you would have to specify:

```
XCOPY V:*.* d:\TAB\DOS5\FINISHED
```

As you can see, you have to spell out the whole path at the target end if you don't want the files just dumped in the root directory. Also, you should definitely add SUBST to the list of utilities that shouldn't be used in conjunction with ASSIGN, BACKUP, DISKCOMP, DISKCOPY, FORMAT, or RESTORE. But used with reasonable care, this one can be a real timesaver.

SWITCHES

Internal to IO.SYS

Function Forces an enhanced (101-key) keyboard to behave like a conventional (84-key) keyboard

Usage Can be used only in the CONFIG.SYS.

Origin Version 4.0.

Format SWITCHES=/k

Switches

/k If you need switches, you need the /k switch.

/w Allows WINA20.386 file to be moved from the root directory and stored on any valid path.

Errorlevel Not supported.

Network use Not supported.

Another of DOS's more obscure commands, you need SWITCHES only if you're using an enhanced keyboard and have some—generally older—software that doesn't correctly interpret the scan codes normally generated by the enhanced keyboard.

I didn't have any software that required SWITCHES, so this is one I wasn't able to check out for myself. According to Microsoft, there's a peculiarity when using using SWITCHES in combination with ANSI.SYS. In that case, you need to put the /k switch on the DEVICE=ANSI.SYS line.

The new /w switch is probably of interest to a far greater number of users. With Windows installed, the setup program installs a read-only file named WINA20.386 in the root directory, which isn't where it really should be. This allows you to move it—say to your WIN directory. It's a two-step process, however. You must:

1. Add a SWITCHES=/w command to your CONFIG.SYS file.
2. Add a DEVICE=[d:][path]WINA20.386 command to the enhanced section of your Windows SYSTEM.INI file.

SYS

External (.COM).

Function Transfers system files to create a bootable disk.

Usage Normally executed only from the DOS command line.

Origin Version 1.0 and later.

Format SYS [*d:*][*path*] *d:*

Switches

/? Displays on-screen help including syntax, switches, etc.

Errorlevel Not supported.

This command allows the system files to be transferred in order to make a previously formatted disk bootable. Its use is somewhat limited, however, because, unless the disk was formatted using the /b option to reserve the space required for these files to be bootable, SYS can do the job *only* if the transfer is accomplished before any other files are copied to the disk. Once that's done, the system cannot be transferred until the disk is reformatted—in which case you might as well just use the FORMAT /s switch and be done with it in one operation. Still, there are times when SYS comes in handy.

The Microsoft documentation for SYS is a bit ambiguous because it seems to indicate that both the source and target drives must be specified in the command. This isn't necessarily so. Specifying a source drive and path is required only if the system files aren't on the default drive and directory, or in the path. If the system files are available on the default drive and directory or in the path, the command is simply:

SYS *d:*

In addition to installing the hidden system files, SYS also copies COMMAND .COM to the target disk.

TIME

Internal to COMMAND.COM

Function Displays the system time or sets computer's internal clock.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0.

Format TIME (displays current time setting, prompts for new input)
or TIME *hh:mm*[*:ss*[*:cc*]]

Switches

/? Displays limited on-screen help.

Errorlevel Not supported.

Network use Supported.

About the only time you have to give much thought to TIME with most of today's computers is when you have to move the hour ahead or back for daylight savings. If even then—I don't think I've bothered changing my laptop for a year or so.

Time- and date-keeping is really just a fringe benefit. Even though computers don't care what time it is or what day or week or even year it is, a computer has to have a clock. And although computers don't take coffee breaks, their most critical internal functions are controlled—scheduled if you will—by their internal clock. So many seconds for this, so many seconds for that, and so on. Synchronizing the internal clock to the real world time and date is a relatively simple matter; keeping it synchronized, then, is just a matter of adding a small battery to keep the clock alive, even when the computer is turned off.

Keeping track of file creation and modification times and dates is probably the most important use you can make of the TIME and DATE functions of your computer. When transferring files from one computer to another, you should never put implicit trust in the time/date stamp of any critical files you're importing, especially if you're overwriting files with earlier time stamps. Computer clocks, like any other clock, aren't infallible and, while working on this book, one of my machines mysteriously lost something like 53 hours, throwing time stamps on new or modified files off by over two days. Which could have been a very costly "lost weekend" if I hadn't caught the discrepancy.

You can also make TIME a part of our computer prompt if you like. The command PROMPT $T will replace your normal prompt, typically DOS's default C>, with something like 19:28:13.06. By itself, that's hardly an improvement, but teamed with some of the other PROMPT options, many people find it useful. One drawback, however, is that, like the time displayed in response to the TIME command, the time displayed as part of a prompt doesn't update while it's displayed but rather continues to show what time it was when you hit the Enter key.

TREE

External (.COM)

Function Displays directory paths on specified drive.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0 and later.

Format TREE *d:*[*path*][*switch*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/a Uses text characters rather than graphics characters to show lines linking subdirectories. Use this switch with printers and/or code pages that don't support graphic characters.

/f Lists all filenames in directories.

Errorlevel Not supported.

Network use Supported.

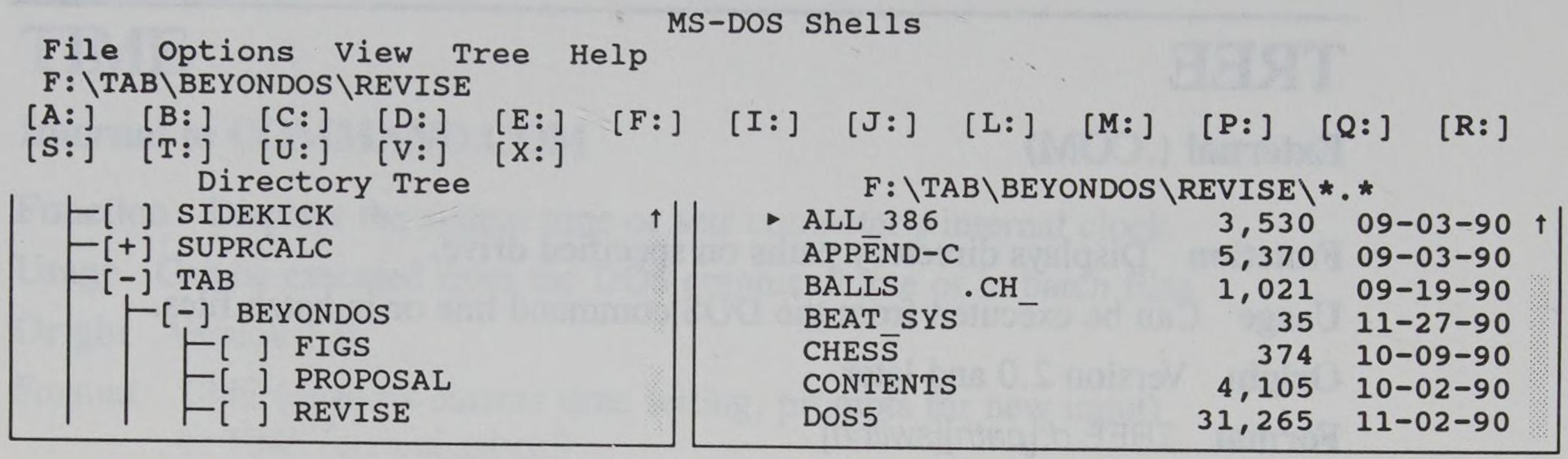
On-screen help Yes

Without the /f switch, TREE seems to be of little practical value. With it, TREE becomes much more useful, but it still pales alongside the way DOSSHELL not only displays your directory structure, but also provides access to your files. The usefulness of TREE would seem to be further eroded by the tremendous enhancement of the DIR command beginning with version 5.0.

```
D:.
├───BETA
├───TAB
│   ├───BEYONDOS
│   └───DOS5
├───TEMP
├───DOS
├───XYWRITE
│   ├───FONTS
│   ├───PRINTERS
│   └───UTILITY
└───ASKSAM
```

```
D:.
    SWAPMAP.EXE
    AUTOEXEC.BAT
    CONFIG.SYS
    MM.DAT
│
├───BETA
│       DUMPS
│
└───TAB
    └───DOS5
            BATCH
            BORROWED
            CATBLURB
```

20-26 Comparison of TREE displays *with* (right) and *without* (left) the /f switch to display filenames.



20-27 Excerpt from a DOSSHELL file list display, which automatically displays both directories and files in alphabetical order, and also allows for scrolling up and down, direct action on selected files, and marking files for group processing.

This isn't meant to detract in any way from the desirability of using a TREE-like display to visualize your directory structure and the contents of included directories.

TREE listings aren't sorted alphabetically, but rather appear in the order you created them (at their particular level), as shown in Fig. 20-26. For comparison, I did a screen capture of a roughly similar directory tree, as displayed by the new DOS 5.0 shell (DOSSHELL—not to be confused with the SHELL command used in the CONFIG.SYS) and shown in Fig. 20-27. Because it's an alphabetical presentation, it obviously shows things in a different order than the TREE display. But the more important difference is the split windowing technique that displays the appropriate file list in the right hand window while you scroll through the directory window.

This isn't meant to try to sell the virtues of SHELL. The SHELL, particularly in graphics mode, is relatively slow, while TREE is considerably faster by comparison.

TRUENAME

Internal to COMMAND.COM
Undocumented

Function Displays the actual drive and path of file(s) accessed.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 4.0.

Format TRUENAME [*d:*][*path*][*filename*[*.ext*]]

Switches None.

Errorlevel Not supported.

Network use Not supported.

Still undocumented, but a sleeper that's been hiding in COMMAND.COM since DOS 4.0, TRUENAME cuts through whatever twisted, convoluted routings you create with SUBST, ASSIGN, and JOIN to tell you where a file is actually located. Why it remains undocumented is anybody's guess—I've never gotten a straight answer from Microsoft. At least for now it's still there if you're interested.

Unlike most commands supported by DOS 5.0, no HELP is provided for it, however, although you really shouldn't need much help with this one. For instance, I use SUBST a lot, especially with word processor files—the batch file that loads my word processor starts by setting up a substitution list, in fact. For the path to all the text files for this book, I substituted V:—because it's the Roman numeral for 5, as in DOS 5.0. So where's V:? Well, if I issue the command:

```
TRUENAME V:
```

TRUENAME returns:

```
D:\TAB\DOS5
```

If your current default happens to be substituted, joined, or assigned, just entering TRUENAME with no parameters will tell you where you really are, despite what the DOS prompt might be telling you (if you've used PROMPT $P to customize your prompt). It doesn't get much easier.

Still, TRUENAME really is of rather limited use. It won't, for instance, help you find a file you've misplaced. It can, however, be a handy troubleshooting aid, so I've included TRUENAME here. For whatever it's worth.

TYPE

Internal to COMMAND.COM

Function Displays contents of ASCII (text) files.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 1.0.

Format TYPE [*d:*][*path*]*filename*[*.ext*]

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Not supported.

TYPE is another old-timer, dating to a time before you really had much else to work with when it came to looking at files. It still comes in handy, although little or nothing has been added to it to enhance it though the years. It still does the same job in the same way it always has.

TYPE is a lot like PRINT in many ways. In fact, if you want to preview the way PRINT will print out a file, you can TYPE it first. If the file itself is formatted with margins, etc., and will PRINT that way, TYPE will display it that way on screen. And, conversely, if you don't like the way the TYPE command displays a file on screen, you probably won't like the way PRINT prints it, either. About the only difference you'll see with TYPE is funny little characters, generally the ASCII 12 (♀), that PRINT interprets as end-of-page marks.

When displaying long files—any file requiring more than one screen—TYPE is of little use, unless you combine it with MORE, as shown below:

TYPE [*d:*][*path*][*filename*[*.ext*]] | MORE

That's actually the hard way to do it, though, because MORE can do the same job all by itself and save you some typing if you do it this way:

MORE <[*d:*][*path*][*filename*[*.ext*]]

Another drawback to using TYPE with longer files (and MORE too, for that matter) is that it scrolls in only one direction. To back up, you've got to go to the end of the file (or Ctrl−C to exit) and start down from the top again—a real nuisance. However, while nothing has ever been done to really enhance TYPE, DOS 5.0 does offer a way to overcome the problem with its new EDIT feature. Not only does EDIT let you scroll back and forth to your heart's content, but lets you edit too. The syntax is simply:

EDIT [*d:*][*path*][*filename*[*.ext*]]

Also, TYPE is suitable only for use in displaying ASCII files. It can't satisfactorily display most spreadsheet, database, or other specially formatted files. It will *try* to display binary files (programs and the like), usually with much beeping but little success—DEBUG is the way to look at them.

So for binary files use DEBUG, and for text you're probably better off with MORE or EDIT. Put in that perspective, TYPE actually serves very little purpose in today's DOS working environment, with the exception of taking a quick look at short files or previewing files you think you want to PRINT to be sure they're properly formatted.

UNDELETE

External (.EXE)

NEW!

Function Restores files deleted with the DEL or ERASE commands.

Usage Should be executed only from the DOS command line.

Origin PC Tools.

Format UNDELETE [*d:*][*path*][*switch(es)*]

Switches

/? Displays on-screen help including syntax, switches, etc.

/list Lists deleted files that may be recovered in specified path.

/dt Recovers only files listed in delete tracing files produced by MIRROR, prompting for confirmation before each file is actually restored.

/dos Ignores delete tracking file if available, seeking directory entries with first character of filename discarded as sole criterion.

/all Recovers all deleted files using delete tracking file if available first, then any deleted files as well, substituting a pound sign (#) for the discarded first character of those filenames (or other character should this result in filename duplication).

Errorlevel Not supported.

Network use Not supported.

UNDELETE is more powerful than most of the current crop of 3rd-party unerase utilities (except for PC Tools from which it's licensed). To realize its full potential, it should be used not just as a stand-alone function, but in conjunction with MIRROR in its TSR mode (with the /t*drive*[-*entries*] switch).

MIRROR as a TSR creates a system (+S) file called PCTRACKR.DEL, which contains data from about every file you erase, including complete filenames. This is what UNDELETE looks for first (unless you use the /dos switch). Restoration of these files—all of them if you've used the /all switch, or selectively with the /dt switch—is completely automatic, including even the first character of every filename that's normally discarded by DOS.

The system seems about as nearly bulletproof as anyone could want. I copied COMMAND.COM to a disk that MIRROR was tracking, then erased that file and copied another version of COMMAND.COM to the disk to see what the following UNDELETE command would do:

```
UNDELETE B: /all
```

and saw the information shown in Fig. 20-28 on screen. Obviously, I chose to

```
Directory: B:\
File Specs: *.*

    Delete Tracking file contains    1 deleted files.
    Of those,    1 files have all clusters available,
                 0 files have some clusters available,
                 0 files have no clusters available.

    MS-DOS Directory contains    0 deleted files.
    Of those,    0 files may be recovered.

Using the Delete Tracking file.

    COMMAND  COM    45466 10/26/90  5:38a  ...A  Deleted: 12/18/90 10:49a

The filename already exists. Enter a different filename.
Press "F5" to bypass this file.
```

20-28 Typical screen display when UNDELETE is run, if MIRROR with the delete tracking (TSR) feature has previously been run.

skip it, but it illustrates the point. Remember, however, that undeleting files is entirely different from restoring them from a backup disk because you're not copying something—rather restoring something that's already on the disk to active status.

Even if MIRROR hasn't been run recently, UNDELETE can still bring back at least your most recently erased files. But without a valid (or up-to-date) PCTRACKR.DEL file to work with, UNDELETE works pretty much like other unerase utilities.

UNFORMAT

External (.COM)
NEW!

Function Restores a hard disk (or floppy) that has been inadvertently reformatted to its prior condition.

Usage Should be executed only from the DOS command line.

Origin PC Tools.

Format UNFORMAT *drive* [*switch(es)*]
or UNFORMAT /partn

Switches

/? Displays on-screen help including syntax, switches, etc.
/j Verifies that files created by MIRROR have been saved and agree with the system information on the disk. Does not initiate unformatting.
/l When used without the /partn switch, it lists every file and subdirectory found by UNFORMAT.
/p Directs all output to the printer so you have a hard copy record of the entire UNFORMAT process. Use this switch only if you don't want UNFORMAT to use files created by MIRROR.
/test Shows how the disk will be unformatted without actually unformatting. Use only if you don't want UNFORMAT to use files created by MIRROR.
/partn Restores a corrupted partition table (hard disks only). Valid only if MIRROR has been used to create a PARTNSAV.FIL file.

Errorlevel Not supported.

Network use Supported (and, although the documentation indicates network support, UNFORMAT cannot be run over a network).

This is one you can hope you'll never need, but it's awfully nice to have it handy just in case. And, in practice, it's surprisingly easy to use. In the case of a floppy, in response to the command UNFORMAT B:, as shown below, it's virtually foolproof:

Insert diskette to rebuild in drive B:
and press ENTER when ready.

Restores the system area of your disk with
the image file created by MIRROR.

WARNING !! WARNING !!

This should be used only to recover from the inadvertent use

```
of the DOS FORMAT command or the DOS RECOVER command.
Any other use of UNFORMAT may cause you to lose data! Files
modified since the last use of MIRROR may be lost.

The LAST time MIRROR was used was at 14:16 on 01-26-91.

The MIRROR image file has been validated.

Are you SURE you want to update the SYSTEM area
of your drive B (Y/N)?y

The system area of drive B has been rebuilt.

You may need to reboot the system.
```

And, unlike various 3rd-party utilities, the first letter in the name of each recovered file is not lost with the DOS UNFORMAT utility. It recovers everything—provided of course that it's run before you start copying any new files to the disk. Once that happens, as with any other unformatting scheme, all bets are off.

Part of the UNFORMAT screen information mentions "the last use of MIR ROR." This can be misleading. In this case, this was from a floppy disk and MIR ROR was *not* used—and had never been used on that disk. The DOS FORMAT (the default) had saved the data from the FAT and root directory to someplace on the disk known only to DOS. During some of the earlier development stages of DOS 5.0, the FAT and root directory data was saved into an easily spotted hidden file called UNFORMAT.DAT, but this visible file structure was discontinued.

Numerous tests have demonstrated no apparent benefit from actually using MIRROR with floppy disks. Even those containing large numbers of files and multiple directory levels were flawlessly recovered by UNFORMAT. I even tried replacing the entire contents of a floppy after running MIRROR to see if I could fool UNFORMAT into restoring the wrong FAT and root directory information, but it didn't work

Despite the fact that there's little to be gained or lost by using MIRROR with floppy disks, it's recommended that you use MIRROR regularly to protect the FAT and root directory and other hard disk data not only from the possibility of accidental reformatting, but from other hard disk hazards as well.

However, as I've stressed elsewhere in this book but which bears repeating, the new nondestructive FORMAT in DOS 5.0 is entirely different from other nondestructive formatters on the market (such as the one included with Norton Utilities). And those differences are incompatible with the new DOS UNFOR MAT program and vice versa. UNFORMAT *cannot* recover disks that have been reformatted with any other formatter I'm familiar with—including its kissing cousin, PCFORMAT (at least through V6), available from PC Tools.

One of the most important but poorly documented uses of UNFORMAT is in hard-disk recovery, following a "crash." If you've run MIRROR on the disk (and this works only if you have), UNFORMAT can in most cases, barring an acute mechanical failure, restore the disk to its pre-crash condition.

It's critical, here, to understand that UNFORMAT really has nothing to do with formatting. Essentially, it simply rewrites the FAT and root directory, as long as MIRROR has preserved that information. It doesn't matter if you're rewriting them because you've accidentally reformatted, or you've had a disk crash.

In any event, DOS's new unformat capability is a welcome addition to its bag of tricks.

VER

Internal to COMMAND.COM

Function Displays the MS-DOS version number.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format VER

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Not supported.

There isn't much to say about this one. It does just one thing. It tells you what DOS version you're running under—an important bit of information you should have handy any time you call for tech support on any product—hardware or software.

The version number returned, as in the example shown below, is the actual version number, and isn't affected by SETVER entries:

```
VER
MS-DOS Version 5.0
```

Keep in mind that other software you have—spreadsheets, word processors, etc.—usually has version numbers as well. This is another number you'll need if calling for tech support, but of course the DOS VER command won't supply it. It can, however, usually be found on one of the opening screens of the program itself, if not on the distribution disk label.

VERIFY

Internal to COMMAND.COM

Function If set on, it tells MS-DOS to verify that the COPY process was properly completed.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format VERIFY [*switch*]
or VERIFY (to display current setting of the switch)

Switches

/? Displays on-screen help including syntax, switches, etc.
on Sets the VERIFY switch on.
off Sets the VERIFY switch off.

Errorlevel Not supported.

Network use Not supported.

Note that this command doesn't verify the integrity of the files or data itself, but rather the process.

This is one of the most misunderstood and, because it's so widely misunderstood, overused commands in the entire MS-DOS bag of tricks. It does *not* verify the integrity of files or data when you're copying from one disk to another (or another location on the same disk). It simply verifies that DOS has gone through the right motions and that, barring some unexpected problem, the copy operation was successful.

About the only useful piece of information that VERIFY will actually tell you is if data has been written to a bad sector on a disk. This, however, is a far cry from assuring file integrity, and can both lead to a sense of false security and slow down your system.

The VERIFY function can be tacked onto COPY commands with the /v switch, but, again, what it verifies is *not* file integrity. When it's important to assure the integrity of any copied files or data, you should use COMP or FC.

VOL

Internal to COMMAND.COM

Function Displays disk volume label and serial number if they exist.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 2.0.

Format VOL *d:*

Switches

/? Displays minimal on-screen help.

Errorlevel Not supported.

Network use Not supported.

Except to test this one so I could say, "yeah, it works," I've never used VOL. And I do use volume labels—especially on floppy disks. But DOS has better ways of reading them, ways that give a lot more information in the bargain. Like DIR, for instance, which, in response to the command DIR A:, reports:

```
Volume in drive A is SITBACK3
Directory of a:\*.*
```

You get the volume label followed by a directory listing (not shown here) and, as an extra benefit, down at the bottom some important disk statistics:

```
51,712 bytes in 21 file(s)
28,160 bytes free
```

TREE also reports the volume label and serial number, if any, as does CHKDSK, which, in addition, returns some fairly detailed information about the general health and well-being of the disk. Both of these take longer to run, however, particularly the latter, which typically takes from a few seconds for a hard disk to as much as a half minute for some of the higher-density floppy drives. Longer than you probably want to wait unless you specifically want the additional information.

XCOPY

External (.EXE)

Function Provides greatly enhanced file copying capabilities.

Usage Can be executed from the DOS command line or in batch files.

Origin Version 3.2 and later.

Format XCOPY [*d:*][*path*]*filename*[*.ext*]
[*d:*][*path*][*filename*[*.ext*][*switch*]

Switches

/? Displays on-screen help including syntax, switches, etc.
/a Copies only files whose archive bits are set—doesn't change the setting.
/d:*mm-dd-yy* Copies only files created or modified after specified date.
/e Re-creates even empty subdirectories when copying directories.
/m Copies files whose archive bits are set, but clears the attribute.
/p Prompts for confirmation (*filename* (Y/N)?) prior to each copy.
/s Copies subdirectories, recreating tree on target disk if required.
/v Verifies that each copy is performed correctly.
/w Instructs XCOPY to wait for disk change before searching source files.

Errorlevel

0 Copy completed without error.
1 No files found to copy.
2 Copy process terminated by user (Ctrl−C or Ctrl−Break).
4 Invalid drive, syntax error, file or path not found.
5 User abort error reading or writing disk.

Network use Not supported.

On-screen help Yes.

Last in DOS's alphabet, perhaps, but certainly not least, XCOPY has to rank as one of the best DOS utilities yet. You might just as well forget about plain old COPY for all the extra benefits XCOPY offers.

With its power to re-create the entire subdirectory tree structure as it goes, XCOPY is a powerful backup tool with one big advantage over BACKUP—you don't have to RESTORE files made with XCOPY to use them. They're exact, ready-to-use copies just like the originals. Even if you go back to an old disk copied using an older version of XCOPY, it doesn't matter—try restoring files backed up on an old disk with some of the earlier version of BACKUP and you'll see the importance of that one real fast. Files from a BACKUP disk must be restored using RESTORE from a version of DOS that's compatible with the BACKUP's DOS version. If a hard disk has been upgraded to a later DOS version in the

interim, you must boot the system from a floppy to accommodate the proper RESTORE version.

Even used without the whistles and bells, XCOPY has a significant advantage over COPY, reducing copying time considerably where a number of files are being copied at once. The advantage lies in the way XCOPY works. Unlike COPY, which reads one file, copies, and then reads another, etc., XCOPY reads as much as it can stuff into RAM in one read operation, then does a continuous write without all the stopping and starting.

Because of its ability to re-create your entire directory structure (or whatever part of it you want to maintain on copy disks) and its ERRORLEVEL support, XCOPY lends itself to use in batch files or macros as a file-by-file backup tool, as demonstrated below in a typical batch file written around XCOPY and a DOSKEY macro equivalent:

```
\DOS\XCOPY $1 $2 $3
IF NOT ERRORLEVEL 0 ECHO XCOPY NOT SUCCESSFUL
IF NOT ERRORLEVEL 0 \DOS\BATCH\ERR
```

```
DOSKEY XCOPY = \DOS\XCOPY $1 $2 $3 $T IF NOT ERRORLEVEL 0
ECHO XCOPY NOT SUCCESSFUL $T IF NOT ERRORLEVEL 0 \DOS\
BATCH\ERR
```

Note that, although the macro is shown on two lines here, it's actually a single DOS line. There must not be a carriage return anywhere except at the end of the macro, or DOS will interpret it as the end of the command.

The ERR at the end of each of the above is an ERRORLEVEL detection batch file. If a nonzero ERRORLEVEL is detected, it returns that value so you can check to see exactly what went wrong. This, combined with various error messages XCOPY can generate internally, make this a pretty powerful tool.

Neither of these as written, I should point out, will check the ERRORLEVEL in case of a user abort, because that will terminate the batch or macro processing—but then you ought to know that if you hit Ctrl−C or Ctrl−Break.

Index

$ symbol, DOSKEY macros use, 92-93
% numerical variable symbol, 90-92
%variable% string indicator, batch files, 92
: DOSKEY macros and batch files label-preface symbol, 90
< DEBUG pointer, 203
= = comparison operator, 92
@ suppression command, batch files, 92

A

accessing files (*see* also APPEND), 216-219
allocation errors, CHKDSK use, 238
ANSI.SYS, 61, 212-215
  DOSSHELL use, 173
  ECHO use, 280
  MODE use, 343
  PROMPT use, 359
  SWITCHES use, 392
APPEND, 85, 119, 216-219
  BACKUP use, 218, 228
  JOIN use, 320
  PATH use, 349
  RESTORE use, 218
  SET use, 218
archive files, set/disable attributes (*see* also ATTRIB), 222-224
ASCII characters, 202
  load character table, GRAFTABL, 307
ASSIGN, 86, 119, 220-221
  BACKUP use, 221, 228
  DISKCOMP use, 263
  DISKCOPY use, 221
  FORMAT use, 221
  JOIN use, 221, 320
  LABEL use, 221, 322
  PRINT use, 221
  RESTORE use, 221
  SUBST use, 221, 391
  TRUENAME use, 397
ATTRIB, 116, 137-138, 140, 145-147, 222-224
  BACKUP use, 223
  EXPAND use, 289
  REPLACE use, 223, 370
  RMDIR use, 373
  XCOPY use, 223
attributes, file, display and/or modify (*see also* ATTRIB), 222-224
AUTOEXEC.BAT, 51-53, 88
  customized date display, DATE, 249-250
  DIRCMD, 52
  EDIT, 53
  LOADHIGH, 53
  MODE, 53
  PROMPT use, 359
  SET, 52, 63

B

BACKUP, 7, 116, 144, 145, 147, 150-152, 225-228
  APPEND use, 218, 228
  ASSIGN use, 221, 228
  batch file using, 226-228
  COUNTRY.SYS use, 246
  file attributes, ATTRIB use, 223
  IO.SYS use, 372
  JOIN use, 228, 320
  MSDOS.SYS use, 372
  RESTORE use, 225, 371-372
  SUBST use, 228, 391
  wildcard use, 225
  XCOPY use, 408
backward compatibility, xvii
BASIC interpreter (*see also* QBASIC), 29, 185, 360
BASICA (*see also* QBASIC), 29, 191, 196-198, 360
batch files, 9, 22, 24-25, 86, 87-102, 211
  % numerical variable symbol, 90-92
  %variable% string indicator, 92
  : label-preface symbol, 90
  = = comparison operator, 92
  @ suppression command, 92
  AUTOEXEC.BAT, 88
  BACKUP batch file, 226-228
  CALL, 89, 233
  change position of replaceable parameters, SHIFT, 89, 91, 383-384
  command-echo on/off, ECHO, 89, 101-102, 279-280
  comments, REM, 89, 366-367
  compare contents, COMP, 242-243

batch files (*cont.*)
conditional testing, IF, 89, 90, 96-99, 315-316
CONFIG.SYS, 88
DO, 89
DOSKEY macros vs., 99-102
ECHO, 89, 101-102, 279-280
environment settings, 65-66
ERRORLEVEL, 89, 98-99
EXIST, 89
FIND use, 94-95, 100-101
FOR, 89
GOTO, 89, 96, 305-306
IF, 89, 90, 96-99, 315-316
IF EXISTS, 97-98
IF [NOT] ERRORLEVEL, 98-99
IN, 89
jump to subroutines, GOTO, 89, 96, 305-306
LABEL, 96
macro memory use vs., 270
memory management, 102
names/syntax, 95-96
null values, 92-93
PATH use, 58
PAUSE, 89, 352-353
PRINT use, 356
REM, 89, 366-367
replaceable parameters, 93-94
SHIFT, 89, 91, 383-384
suspend execution, PAUSE, 89, 352-353
temporary control transfer, CALL, 89, 233
terminate execution, BREAK, 229-230
variable passing, 91
binary files, EXE-to-COM file conversion (*see also* EXE2BIN), 286
BIOS, 4-5, 11-13
IBMBIO.COM, 16-17
IO.SYS use, 16-17, 318
load high-ASCII character table, GRAFTABL, 307
loading DOS, 15-16
boot sectors, 15, 124
bootable-disk creation, SYS, 393
bootstrap loader, 16
BREAK, 6, 8, 22, 86, 229-230
CONFIG.SYS use, 41-43
IO.SYS support, 318
BUFFERS, 6, 231-232
CONFIG.SYS use, 41, 43-44
FASTOPEN use, 290
FILES use, 296
IO.SYS support, 318
SMARTDRV.SYS use, 44, 232, 386

C

cache memory
BUFFERS use, 232
disk-cache creation, SMARTDRV.SYS, 80-82, 385-386
FASTOPEN use, 290
CALL, 8, 22, 24, 233
batch files, 89
COMMAND.COM use, 241
DOSKEY macros, 89
GOTO use, 306
case conversions, international setting, COUNTRY.SYS, 246
case sensitivity, 210
CHCP, 8, 9, 22, 211, 234
COUNTRY use, 234
KEYB use, 234
MODE use, 234
NLSFUNC use, 234, 348
CHDIR, 8, 22, 235-236
PROMPT use, 236
child processes, environment, 66, 67
CHKDSK, 7, 9, 116, 139, 237-239, 407
clones, 12
CLS, 8, 22
code page settings
console (display) device (*see also* DISPLAY.SYS), 267
display/change (*see also* CHCP), 234
mode setting (*see also* MODE), 337, 342
parallel port switching (*see also* PRINTER.SYS), 357
switch using CHCP (*see also* NLSFUNC), 348
color selection, ANSI.SYS 213-215
COM files, 5, 9, 22, 23, 209
EXE-to-COM file conversion, EXE2BIN, 286
MSDOS.SYS, 347
PATH use, 58-59
COMMAND.COM, 5, 7-9, 10, 21-25, 116, 240-241
batch file processing, 24-25
CALL use, 241
CONFIG.SYS use, 241
DOSKEY macros use, 23-24
error messages, 25
excess files, removal, JUNKYARD files, 115-119
execution of programs, 21
EXIT, 287
FOR use, 299
HELP, 72
internal commands, 22
IO.SYS use, 318
order of operations, 22-23
PATH use, 23, 350
RAMDRIVE.SYS use, 362
SET use, 240, 375
SHELL use, 381-382
SYS use, 393
VER use, 405
commands
case sensitivity, 210
command processors (*see* COMMAND.COM)
command-line editing (*see* DOSKEY macros)
command-processor control, EXIT, 287
DOS shell (*see* DOSSHELL)
drive specifications, 210
entry procedures, 210
execute specific command in specific file, FOR, 299
external, 7-10, 209, 240
filenames, 210
internal, 7, 8, 22, 209, 240
interpreters, 5, 7, 21
non-DOS command processor (*see also* SHELL), 381-382
on-screen help, HELP, 310-312
parameters, 210
paths, 210
recall last DOS command, DOSKEY macros, 269-272
spacing, 210-211
switches, 210
syntax, 210
variables, 210
COMMENT, 6
IO.SYS support, 318
comments in code, REM, 366-367
COMP, 119, 406, 242-243
DISKCOMP use, 263
FC utility use, 291

compatibility issues, xvi-xvii
COMSPEC, 55, 56-58
  RAMDRIVE.SYS use, 362
  SET use, 64, 375
  SHELL use, 382
conditional testing, IF, 315-316
CONFIG.SYS, 5, 8, 37, 39-52, 61, 88, 212
  AUTOEXEC.BAT files, 52-53
  BREAK use, 41, 42-43
  BUFFERS use, 41, 43-44
  COMMAND.COM use, 241
  COPY CON use, 42
  COUNTRY use, 41, 44, 246
  device drivers, DEVICE, 40, 41, 45-46, 257
  DEVICEHIGH use, 41, 47, 258
  disk management, 40-41
  DOS kernel, 47-48, 268
  DOS=HIGH, 47, 48
  DOS=UMB, 48
  DOSSHELL, 173
  DRIVPARM use, 41, 48, 277-278
  EDIT use, 42
  EGA.SYS use, 283
  EMM386, 45-48
  FCBS use, 41, 48, 293
  FILES setting, 41, 48-49, 296
  HELP, 72
  HIMEM.SYS use, 34-35, 45-48, 313-314
  INSTALL use, 41, 49-50, 317
  installation procedures, 33-35
  international setting, COUNTRY.SYS, 41, 44, 246
  IO.SYS use, 318
  LASTDRIVE use, 41, 50, 323
  memory management, 45-48
  RAMDRIVE.SYS, 46
  REM use, 366-367
  SETUP program use, 33-34
  SETVER use, 31-33
  SHARE use, 380
  SHELL use, 41, 50-51, 381-382
  SMARTDRV.SYS, 46
  STACKS use, 41, 51, 389
  SUBST use, 391
  SWITCHES use, 41, 51-52, 392
  VERIFY use, 41, 52
configuration, FDISK, 294-295
coprocessor, on/off (*see also* EMM386), 284
COPY, 7, 8, 10, 22, 132, 143-146, 174, 211, 244-245
  DISKCOPY use, 263-265
  DRIVER.SYS use, 275-276
  HELP, 72
  REPLACE use, 245
  SUBST use, 391
  VERIFY use, 406
  XCOPY use, 244-245, 408
COPY CON, 20
  CONFIG.SYS use, 42
COUNTRY, 6, 117, 246
  BACKUP use, 246
  CHCP use, 234
  CONFIG.SYS use, 41, 44
  DATE use, 246
  DEVICE use, 257
  IO.SYS support, 318
  RESTORE use, 246
  TIME use, 246
CTTY, 8, 22, 247-248
  MODE use, 247
  NUL devices, 247
currency, international setting (*see also* COUNTRY.SYS), 246
cursor/cursor movement (*see also* ANSI.SYS), 212-215

D

DATE, 8, 22, 249-250, 394
  ECHO use, 280
  international setting, COUNTRY.SYS, 246
DEBUG, 199-205, 251-252
  < pointer, 203
  BREAK use, 229-230
  ECHO use, 201, 280
  EDLIN use, 282
  Escape characters, 201-204
  MEM use, 328-330
  scripts, 200, 204
  SETVER use, 378
  TYPE use, 399
decimal numbers, 202
decimal separators, international setting, COUNTRY.SYS, 246
DELETE, 7, 8, 22, 174, 253-254
  UNDELETE use, 400
  UNERASE use, 254
DELOLDOS, 35-36, 255-256
DESQview, 18, 22, 66, 67
  APPEND use, 219
  DOSSHELL, 182
  FILES use, 296
  installation procedures, 38
  JOIN use, 320
  memory management, 107
DEVICE, 6, 107, 111, 257
  CONFIG.SYS use, 41, 45-46
  COUNTRY.SYS use, 257
  EMM386.SYS use, 257
  HIMEM.SYS use, 257
  IO.SYS support, 318
  KEYBOARD.SYS use, 257
  RAMDRIVE.SYS use, 257
  SMARTDRV.SYS use, 257
device drivers, 5, 40, 41, 45-46
  ANSI.SYS, 212-215
  code page switching, console device (*see also* DISPLAY.SYS), 267
code page switching, display/change (*see also* CHCP), 234
COUNTRY.SYS, 246
define parameters for nonstandard devices (*see also* DRIVPARM), 277
DRIVER.SYS, 275-276
EGA.SYS, 283
HIMEM.SYS, 313-314
KEYB, 321
load into memory, DEVICE, 257
load into reserved memory (*see also* DEVICEHIGH), 258
memory management, 110-114
SMARTDRV.SYS, 80-82, 385-386
DEVICEHIGH, xvi, 38, 111-112, 258
  CONFIG.SYS use, 41, 47, 258
  EMM386.SYS use, 258
  HIMEM.SYS use, 258, 314
  IO.SYS support, 318
DIR, xiv, 7, 8, 22, 78, 135-137, 140, 259-261, 407
  DIRCMD, 62-63, 260-262
  MORE use, 345
DIRCMD, 56, 137, 262
  AUTOEXEC.BAT, 52
  DIR use, 62-63, 260-262
  SET, 62-64, 137, 262, 375
directories and subdirectories, 135-142
  alias drive designation (*see also* SUBST), 390-391
  APPEND delimiter use, 216-217
  attributes (*see also* ATTRIB), 137-138, 140

directories and subdirectories (*cont.*)
attributes (*see also* DIR), 259
BACKUP use, 226
check status, CHKDSK, 139, 237-239
create, MKDIR, 335-336
current, display/change, CHDIR, 235-236
DIR switches become default (*see also* DIRCMD), 262
DIR, 135-137, 140
DIRCMD, 137, 262
display actual drive/path of file, TRUENAME, 397
display list, DIR, 259-261
DOSSHELL, 135, 139-141
link drives, JOIN, 320
MOVE, 141
parent directory, 236
paths, PATH, 236, 335, 349-351
quick-access, FASTOPEN, 82, 290
remove directories, RMDIR, 373-374
renamed files, RENAME, 368
retrieve backup files, RESTORE, 371-372
root directory, 236
root, recovery, MIRROR, 331-334
sorting, DIR, 259
sorting, SORT, 387-388
tree structure, TREE, 138-140, 395-396
disk management, 121-134
bad or defective disk recovery, RECOVER, 363-365
batch files use, 102
boot sectors, 124
capacity of disks, 124-128
check memory/status, CHKDSK, 237-239
CONFIG.SYS, 40-41
configure hard disk for MS-DOS, FDISK, 294-295
COPY, 132
directories and subdirectories, 135-142
DISKCOPY, 124, 132-133
DOSKEY macros use, 102
file allocation table (FAT), 124, 237
file system, DOS kernel, 19-20
floppy disks, compare, DISKCOMP, 263
floppy disks, copy , DISKCOPY, 264-266
format-error recovery, UNFORMAT, 129-134, 402-404
formatting, 124-128
formatting, FORMAT, 126-128, 301-304
fragmentation, 128-134, 238-239
media compatibility, 123-124
MIRROR, 128, 134, 331-334
partitioning, 238
restore disk after format error, MIRROR, 128, 134, 331-334
root directories, 124
tracks, 124
UNDELETE, 129, 134
UNFORMAT, 134, 402-404
virtual disk in RAM, RAMDRIVE.SYS, 83-85, 238, 361-362
volume label, create, LABEL, 322
volume label, display, VOL, 407
XCOPY, 132
DISKCOMP, 119, 263
ASSIGN use, 263
COMP use, 263
COPY use, 263
FC utility use, 263
JOIN use, 263, 320
SUBST use, 263, 391
XCOPY use, 263
DISKCOPY, 116, 124, 132-133, 264-266
ASSIGN use, 221
COPY use, 264-265
DRIVER.SYS use, 275-276
JOIN use, 320
REFORMAT use, 265
REPLACE use, 265
SUBST use, 391
XCOPY use, 264-265
display adapters, 267
display screens
23-line scrolling output, MORE, 345-346
ANSI.SYS, 212-215
color selections, ANSI.SYS, 213-215
command displayed on screen, ECHO, 279-280
EGA monitors, EGA.SYS, 283
load high-ASCII character table, GRAFTABL, 307
mode setting, MODE, 337, 338, 343
prompt, customized, ANSI.SYS, 213-214
resolution setting, DOSSHELL, 273
DISPLAY.SYS, 117, 267
DO, batch files and DOSKEY macros, 89
DOS command, 268
CONFIG.SYS use, 268
IO.SYS support, 318
LOADHIGH use, 268
DOS extenders, 104
DOS=HIGH, xv, 6, 35, 37, 47, 48, 107, 111
DOS=LOW, 6
DOS=NOUMB, 6
DOS=UMB, xv, 6, 48, 111
DOSHELP.HLP, 71, 74-75, 311-312
DOSKEY macros, xiv, 87-102, 116, 150, 211, 269-272
$ symbol use, 92-93
: label-preface symbol, 90
== comparison operator, 92
batch files vs., 99-102, 270
CALL, 89
COMMAND.COM use, 23-24
DO, 89
ECHO, 89, 101-102
environment, 65
ERRORLEVEL, 89, 98-99
EXIST, 89
FC utility use, 292
FIND use, 94-95,100-101, 298
FOR, 89
FORMAT use, 303
GOTO, 89, 96
IF, 89, 90, 96-99
IF EXISTS, 97-98
IF [NOT] ERRORLEVEL, 98-99
IN, 89
LABEL, 96
memory management, 102
names/syntax, 95-96
PATH use, 350
PAUSE, 89
PRINT use, 356
recall DOS command, DOSKEY macros, 269-272

REM, 89
replaceable parameters, 93-94
SET use, 375
SHIFT, 89
XCOPY use, 409
DOSSHELL, xiv, 78, 118, 135, 139-141, 171-184, 273
ANSI.SYS, 173
CONFIG.SYS, 173
default settings, custom, 178-180
DESQview, 182
DOSSWAP use, 273, 274
EGA.SYS use, 283
file management, 174-175
FORMAT use, 304
HELP use, 71, 312
HIMEM.SYS, 173
installation procedures, 32
JOIN use, 320
Main menu window, 176-178
mode setting, MODE, 173
MOVE, 175-176
QBASIC, 173
RAMDRIVE.SYS, 183
recovering lost data, 183-184
task switching, 180-183
TREE use, 395
Windows, 182
XTREE vs., 174
DOSSWAP, DOSSHELL use, 273, 274
drive designations, 210
aliasing, SUBST, 390-391
define parameters for nonstandard device, DRIVPARM, 277
link drives, JOIN, 320
maximum-number setting, LASTDRIVE, 323
reassign disk designation, ASSIGN, 220-221
reassign disk designation, SUBST, 220-221
use unsupported physical disk, DRIVER.SYS, 275-276
DRIVER.SYS, 275-276
DRIVPARM, 6, 277-278
CONFIG.SYS use, 41, 48, 276-277
DRIVER.SYS use, 276
IO.SYS support, 318
DV.COM, 22

E

ECHO, 8, 22, 24, 279-280
ANSI.SYS use, 280
batch files, 89, 101-102
DATE use, 280
DEBUG use, 201, 280
DOSKEY macros, 89, 101-102
EDIT use, 280
ERRORLEVEL use, 279
MORE use, 280
EDIT, xiv, xvii, 24, 116, 118, 185-190, 281
AUTOEXEC.BAT, 53
CONFIG.SYS use, 42
ECHO use, 280
EDLIN use, 281, 282
HELP use, 71, 312
key commands, 188-189
shortcut keys, 188
TYPE use, 398, 399
EDLIN, xiv, xvii, 86, 119, 185, 190, 282
DEBUG use, 282
EDIT use, 281, 282
EGA.SYS, 283
EMM386, xv, 38, 79-80, 104-105, 110, 113-114, 116, 284-285
CONFIG.SYS use, 45-48, 284
DEVICE use, 257
DEVICEHIGH use, 258
HIMEM.SYS use, 284
MEM use, 328-330
environment, 55-68
ANSI.SYS, 61, 212-215
batch files, 65-66
child processes, 66, 67
COMSPEC, 55-58
CONFIG.SYS, 61
DIRCMD, 56, 62-65
DOSKEY macros use, 65
multitasking, 66, 67
parent processes, 66, 67
PATH, 55-56, 58-60, 66, 67
PROMPT, 56, 60-62
SET, 56, 62-66, 67
SHELL, 56, 57
ERASE, 8, 22, 253, 400
error messages, COMMAND.COM, 25
ERRORLEVEL, 8, 22, 24
batch files, 89, 98-99
DOSKEY macros, 89, 98-99
ECHO use, 279
HELP, 75
PAUSE use, 352
PRINT use, 356
REPLACE use, 370
RESTORE use, 371-372
test, GOTO, 305
XCOPY use, 409
Escape character vs. ESC key, 201-204, 213-214
EXE files, 9, 22, 23, 209
EXE-to-COM file conversion, EXE2BIN, 286
PATH use, 58-59
set version numbers, SETVER, 377-379
EXE2BIN, 119, 286
EXIST, 8, 22, 24, 89
EXIT, 8, 22, 287
EXPAND, 116, 119, 288-289
ATTRIB use, 289
FORMAT use, 289
installation procedures, 33
SETUP program use, 288-289
expanded memory (*see also* memory management), 111
extended memory (*see also* memory management), 34-35, 104, 106
extenders, DOS, 104
extensions, 209
external commands, 7-10, 209, 240

F

FASTOPEN, 37, 82, 119, 290
BUFFERS use, 290
INSTALL use, 49, 317
MKDIR use, 336
SMARTDRV.SYS use, 290
FC utility, 116, 243, 291-292, 406
COMP use, 291
DISKCOMP use, 263
DOSKEY macros inclusion, 292
FCBS, 6, 293
CONFIG.SYS use, 41, 48, 293
IO.SYS support, 318
FDISK, 119, 294-295
file allocation table (FAT), 124
check status, CHKDSK, 237-239
recovery (*see also* MIRROR), 164-166, 331-334
UNFORMAT, 402-404
file operations
accessable to calls, FILES, 296
accessing files, nonexecutable, APPEND, 216-219

file operations (*cont.*)
append files, COPY, 244-245
ASCII files, create/change, EDIT, 281
attributes, set/disable, ATTRIB, 222-224
backup files, BACKUP, 225-228
compare contents, COMP, 242-243
compare contents, FC utility, 243, 291-292
compare floppy disks, DISKCOMP, 263
copy all files on floppy, DISKCOPY, 264-266
copy (*see also* COPY), 244-245
copying files (*see also* XCOPY), 408-409
create=change, EDLIN, 282
debugging, DEBUG, 251-252
delete old DOS files, DELOLDOS, 255-256
delete, DEL, 253-254
deleteing unnecessary COMMAND.COM files, 115-119
directory listing, DIR, 259-261
display actual drive/path, TRUENAME, 397
display contents, TYPE, 398-399
erasing files, Hidden and System attributes, 224-225,
EXE-to-COM file conversion, EXE2BIN, 286
executable files, set version numbers, SETVER, 377-379
execute specific command in specific file, FOR, 299
quick-access, FASTOPEN, 82, 290
recovery of lost files, MIRROR, 331-334
recovery of lost files, RECOVER, 363-365
renaming files, RENAME, 368
replace files/add selective files, REPLACE, 369-370
restore deleted/erased files, UNDELETE, 400-401
retrieve backup files, RESTORE, 371-372
retrieve erased file, UNERASE, 254
search for string, FIND, 297-298
sharing files, SHARE, 380
sharing, control blocks, FCBS, 293
sort data, SORT, 387-388
system files, bootable-disk creation, SYS, 393
trace directory/subdirectory path, PATH, 349-351
verify COPY, VERIFY, 406
file system, DOS kernel, 19-20
filenames, 210
FILES, 6, 148, 296
BUFFERS use, 296
CONFIG.SYS use, 41, 48-49, 296
IO.SYS support, 318
FIND, 116, 297-298, 387
batch files use, 94-95, 100-101
DOSKEY macros use, 94-95, 100-101, 298
FOR use, 297-299
wildcard use, 211, 297
floppy disks
compare disks (*see also* DISKCOMP), 263
DISKCOPY, 264-266
formatting (*see also* FORMAT), 301-304
soft sectoring, 20
FOR, 8, 22, 24, 299-300
batch files, 89
COMMAND.COM use, 299
DOSKEY macros, 89
FIND use, 297-299
REPLACE use, 300
XCOPY use, 300
form factor switch, FORMAT, 303
FORMAT, xiii, xv, xvii, 7, 9, 19, 37, 77, 116, 126-128, 153, 157-159, 301-304
ASSIGN use, 221
DOSKEY macros use, 303
DOSSHELL use, 304
EXPAND use, 289
form factor switch, 303
JOIN use, 320
MIRROR use, 334
SUBST use, 391
SYS use, 393
UNDELETE use, 302-303
UNFORMAT use, 153, 302-303, 402-404
formatting, 19, 124-128, 157-159
format-error recovery, UNFORMAT, 402-404
LABEL use, 322
restore disk after format error, MIRROR, 128, 331-334
UNDELETE, 129
fragmentation, 128-134, 238-239

G

GORILLA.BAS, 119
GOTO, 8, 22, 24, 89, 96, 305-306
batch files, 96
CALL use, 306
DOSKEY macros, 96
ERRORLEVEL use, 305
IF use, 305
GRAFTABL, 307
GRAPHICS, 116, 308-309
INSTALL, 49
load high-ASCII character table, GRAFTABL, 307
graphics adapters, mode setting, MODE, 337
GWBASIC (*see also* QBASIC), xii, xiv, 191, 196-198, 360

H

HELP, xiv, 71-76, 310-312
COMMAND.COM, 72
CONFIG.SYS, 72
COPY, 72
DOSHELP.HLP use, 74-75, 311-312
DOSSHELL use, 312
EDIT use, 312
ERRORLEVEL, 75
INSTALL, 72
QBASIC use, 193-196, 312
REPLACE, 72
SHELL, 72
XCOPY, 72
hexadecimal numbers, 202
hidden files
MSDOS.SYS, 347
set/disable attributes, ATTRIB, 222-224
system, IO.SYS, 318-319
high memory area (HMA) (*see* HIMEM.SYS; memory management)
HIMEM.SYS, xv, 79-80, 104-109, 114, 116, 118, 173, 313-314

CONFIG.SYS use, 34-35, 45-48, 313-314
DEVICE use, 257
DEVICEHIGH use, 258, 314
EMM386 use, 284
installation procedures, 37-38
LOADFIX use, 324
MEM use, 327-330

I

IBMBIO.COM, 5, 7, 16
IBMDOS.COM, 5, 7, 16
IF, 8, 22, 24, 89, 315-316
batch files, 89, 90, 96-99
DOSKEY macros, 89, 90, 96-99
GOTO use, 305
IF EXISTS, batch files and DOSKEY macros, 97-98
IF [NOT] ERRORLEVEL, 98-99
IN, batch files and DOSKEY macros, 89
INSTALL, xvi, 6, 77, 317
CONFIG.SYS use, 41, 49-50, 317
FASTOPEN use, 49, 317
GRAPHICS, 49
HELP, 72
IO.SYS support, 318
KEYB use, 49, 317, 321
NLSFUNC use, 49, 317
SHARE use, 49, 317, 380
installation, 29-38
CONFIG.SYS file, 33-35, 37
delete old DOS files, DELOLDOS, 35-36, 255-256
DESQview, 38
DOS=HIGH option, 37
DOSSHELL loading, 32
EMM386.EXE, 38
expand MS-DOS files from distribution disks (*see also* EXPAND), 33, 288-289
FASTOPEN, 37
HIMEM.SYS, 37-38
KEYB use, 321
LOADFIX, 38
network application, 37
OEM version, 36-37
SETUP program, 29, 31-33
SETVER, 31-33
SHARE use, 380
TSR programs, load, INSTALL, 317
UNINSTALL programs, 30-31
upgrading from earlier DOS versions, 30-31
internal commands, 7, 8, 22, 209, 240
interpreters (*see also* QBASIC), 5, 7, 21, 185, 191, 360
interrupts, STACKS, 389
IO.SYS, 5-8, 13-14, 16, 25, 47, 318-319
BACKUP use, 372
MSDOS.SYS, 347

J

JOIN, 86, 119, 320
APPEND use, 320
ASSIGN use, 221, 320
BACKUP use, 228, 320
DESQview use, 320
DISKCOMP use, 263, 320
DISKCOPY use, 320
DOSSHELL use, 320
FORMAT use, 320
RESTORE use, 320
SUBST use, 320
TRUENAME use, 397
XTREE Pro Gold use, 320
JUNKYARD files, 115-119

K

kernel, DOS, 5, 17-20
CONFIG.SYS, 47-48
file system, 19-20
memory management, 18-19
peripheral support, 19
process control, 18
KEYB, 117, 321
CHCP use, 234
INSTALL use, 49, 317, 321
KEYBOARD.SYS, DEVICE use, 257
keyboard management
ANSI.SYS, 212-215
enhanced-to-conventional performance, SWITCHES, 392
KEYB use, 117, 321
reassignment of keys, ANSI.SYS, 212-215
speed, typematic rate setting, MODE, 344

L

LABEL, 96, 116, 322
ASSIGN use, 221, 322
batch files, 96
DOSKEY macros, 96
SUBST use, 322
label-preface (:), batch files and DOSKEY macros, 90
LASTDRIVE, 6, 323
CONFIG.SYS use, 41, 50, 323
IO.SYS support, 318
SUBST use, 50, 323, 391
LH, 8, 22
loader, DOS, BIOS, 15-16
LOADFIX, xvi, 324
HIMEM.SYS use, 324
installation procedures, 38
LOADHIGH, xvi, 8, 22, 38, 325-326
AUTOEXEC.BAT, 53
DOS command use, 268

M

macros (*see* DOSKEY macros)
MD (*see* MKDIR)
MEM, xvi, 107, 116, 327-330
DEBUG use, 328-330
EMM386.EXE use, 328-330
HIMEM.SYS use, 327-330
MORE use, 328-330
MEMO, SET use, 63
memory management, xiv, 103-114
allocate RAM buffers (*see also* BUFFERS), 231-232
batch files use, 102
cache memory (*see also* BUFFERS), 232
CHKDSK use, 237-239
CONFIG.SYS, 45-48
DESQview, 107
device driver load to reserved memory, DEVICEHIGH, 111-112, 258
device driver loaded to memory, DEVICE, 107, 111, 257
device drivers, 110-114
disk-cache creation, SMARTDRV.SYS, 80-82, 385-386
display memory-utilization information, MEM, 327-330
DOS extenders, 104
DOS kernel, 18-19
DOS=HIGH, 107, 111
DOS=UMB, 111
DOSKEY macros use, 102
EMM386, 104-105, 110, 113-114

memory management (*cont.*)
expanded memory, 111
expanded memory emulation on/off (*see also* EMM386), 284
extended memory use, HIMEM.SYS, 34-35, 104-109, 313-314
FASTOPEN use, 290
load program to reserved memory, LOADHIGH, 325-326
load programs to low memory, LOADFIX, 324
maintain MS-DOS-to-reserved memory link, DOS command, 268
MEM, 107
RAMDRIVE.SYS, 109, 110
read-only memory (ROM), 4-5, 13, 17
REM, 107
reserved (high) memory, 6-8, 37-38, 79-80, 104-109
SHARE use, 380
SMARTDRV.SYS, 109, 110
stack frame allocation for hardware interrupts, STACKS, 389
virtual disk in RAM creation, RAMDRIVE.SYS, 83-85, 361-362
XMA2EMS.SYS and XMAEM.SYS drivers, 110
messages, display on-screen, ECHO, 279-280
MIRROR, xv, xviii, 116, 128, 134, 150, 153-157, 159-168, 331-334
FORMAT use, 334
partition data recovery, 333-334
REBUILD use, 331-334
UNDELETE use, 331-334, 400-401
UNFORMAT use, 331-334, 403
MKDIR, 8, 22, 335-336
MODE, 116, 337-344
ANSI.SYS use, 343
AUTOEXEC.BAT, 53
CHCP use, 234
code page settings, 342
CTTY use, 247
display device status, 338
DOSSHELL, 173
keyboard typematic rate, 344
parallel printer configuration, 338-342
parity bits, 341
redirect output to serial port, 341-342
serial port configuration, 339-341
stop bits, 341
transmission rates, 340
modems, mode setting, MODE, 339-341
MONEY.BAS, 119
MORE, 116, 345-346, 387
DIR use, 345
ECHO use, 280
MEM use, 328-330
PRINT use, 355
SORT use, 345
TYPE use, 345, 398
MOVE, 141, 174-176
MSDOS.SYS, 5, 7, 13-14, 16, 347
BACKUP use, 372
IO.SYS, 347
SETVER use, 378
multitasking (*see also* DESQview; task switching; Windows), 18, 66, 67, 180, 296
MULTITRACK, 6

N

networking, xvi
installation procedures, 37
share, lock, etc., SHARE, 380
xvi, 410
NIBBLES.BAS, 119
NLSFUNC, 37, 117, 211, 348
CHCP use, 234, 348
INSTALL use, 49, 317
nonexecutable files, accessing, APPEND, 216-219
NOT, 8, 22
NUL devices, CTTY use, 247
numerical variables (%), batch files, 90-92

O

OEM DOS 5.0 version, installation, 36-37
operating systems, 3-5, 13-14, 20
OS/2, 13

P

PACKING.LST, 119
parameters, 210
parent directory, 236
parent process, environment, 66, 67
parity bits, MODE, 341
partitioning, 238
recovery after format error, MIRROR, 331-334
passwords, SET, 65, 375-376
PATH, 8, 22, 23, 55-56, 58-60, 66, 67, 210, 216, 217, 218, 236, 250, 335, 349-351
APPEND use, 349
COMMAND.COM use, 23, 350
DOSKEY macros use, 350
SET, 63-65
tree structure, TREE, 395-396
PAUSE, 8, 22, 24, 352-353
batch files, 89
DOSKEY macros, 89
ERRORLEVEL use, 352
peripheral support, DOS kernel, 19
piping
MORE use, 345
SORT use, 388
PRINT, 116, 354-356
ASSIGN use, 221
batch files use, 356
DOSKEY macros use, 356
ERRORLEVEL use, 356
MORE use, 355
TYPE use, 355, 398
PRINTER.SYS, 357
printing
mode setting, MODE, 337-342
print text files, PRINT, 354-355
printer code page setting, parallel port, PRINTER.SYS, 357
send screen display to printer,GRAPHICS, 308-309
process control, DOS kernel, 18
programs, 3
PROMPT, 8, 22, 56, 60-62, 358-359, 394
AUTOEXEC.BAT use, 359
CHDIR use, 236
customizing prompts, ANSI.SYS, 213-214, 359
SET, 64

Q

QBASIC, xiii, xiv, 29, 116, 118, 185, 190-198, 360
  BASICA-program conversion, 196-198
  DOSSHELL, 173
  GWBASIC-program conversion, 196-198
  HELP use, 71, 193-196, 312
  unsupported keywords, 196-197
QRAM memory manager, 35
QuickBASIC, 191, 197

R

RAMDRIVE.SYS, 83-85, 109, 116, 118, 361-362
  COMMAND.COM use, 362
  COMSPEC use, 362
  CONFIG.SYS, 46
  DEVICE use, 257
  DOSSHELL, 183
  SMARTDRV.SYS use, 83
RD (*see* RMDIR)
read-only files, set/disable attributes, ATTRIB, 222-224
read-only memory (ROM), 4-5, 13, 17
README.TXT, 119
REBUILD, MIRROR use, 331-334
RECOVER, 116, 162, 363-365
recovery of lost data, 153-160
  bad or defective disks, RECOVER, 162, 363-365
  DOSSHELL, 183-184
  FORMAT revision, 157-159
  format-error recovery, MIRROR, 153-157, 159-168, 331-334
  format-error recovery, UNFORMAT, 153, 160-168, 402-404
  restore deleted/erased files, UNDELETE, 153-157, 160, 400-401
  tracking erased files, 332-333
redirection of I/O, 20
REFORMAT, DISKCOPY use, 265
REM, 6, 8, 22, 24, 107, 366-367
  batch files, 89
  CONFIG.SYS use, 366-367
  DOSKEY macros, 89
  IO.SYS support, 318
REMLINE.BAS, 119
REN (*see* RENAME)
RENAME, 8, 22, 368
REPLACE, 7, 116, 144, 145, 149, 211, 369-370
  ATTRIB use, 370
  COPY use, 245
  DISKCOPY use, 265
  ERRORLEVEL use, 370
  file attributes, ATTRIB use, 223
  FOR use, 300
  HELP, 72
  SUBST use, 391
  wildcard use, 147-148
  XCOPY use, 370
replaceable parameters, batch files and DOSKEY macros, 93-94
reserved memory (*see* memory management)
resolution setting, DOSSHELL, 273
RESTORE, 116, 150-152, 371-372
  APPEND use, 218
  ASSIGN use, 221
  BACKUP use, 225, 371-372
  COUNTRY.SYS use, 246
  ERRORLEVEL use, 371-372
  JOIN use, 320
  SUBST use, 391
  XCOPY use, 408
RMDIR, 8, 22, 373-374
  ATTRIB use, 373
ROM BIOS (*see* BIOS)
root directory, 124, 236

S

scripts, DEBUG, 200, 204
search/search-and-replace, FIND, 297-298
SELECT program (*see* SETUP program)
SET, 8, 22, 56, 62-65, 250, 375-376
  APPEND use, 218
  AUTOEXEC.BAT, 52, 63
  COMMAND.COM use, 240, 375
  COMSPEC use, 64, 375
  DIRCMD use, 62-65, 137, 262, 375
  DOSKEY macros use, 375
  MEMO use, 63
  multitasking environment, 66, 67
  PATH, 63-65
  PROMPT, 64
SETUP program, 29, 31-33
  CONFIG.SYS use, 33-34
  DELOLDOS use, 35-36, 255
  EXPAND use, 288-289
  high memory area (HMA) use, 37-38
  OEM version, 36-37
SETVER, xvi, 116, 377-379
  CONFIG.SYS use, 31-33
  DEBUG use, 378
  installation, 31-32
  MSDOS.SYS use, 378
  VER use, 405
SHARE, 119, 380
  CONFIG.SYS use, 380
  INSTALL use, 49, 380
shared files, control-block setup, FCBS, 293
SHELL, 6, 9, 56, 57
  COMMAND.COM use, 381-382
  COMSPEC use, 382
  CONFIG.SYS use, 41, 50-51, 381-382
  HELP, 72
  IO.SYS support, 318
  TREE use, 396
shell to DOS (*see also* DOSSHELL), 273
SHELL (*see also* DOSSHELL), 381-382
SHIFT, 8, 22, 24, 89, 383-384
  batch files, 89, 91
  DOSKEY macros, 89
SMARTDRV.SYS, 80-82, 109, 116, 118, 385-386
  BUFFERS use, 44, 232, 386
  CONFIG.SYS, 46
  DEVICE use, 257
  FASTOPEN use, 290
  MKDIR use, 336
  RAMDRIVE.SYS use, 83
SORT, 119, 387-388
  MORE use, 345
source disk, copying, 264
spacing in command entry, 210-211
STACKS, 6, 389
  CONFIG.SYS use, 41, 51, 389
  IO.SYS support, 318

stop bits, MODE, 341
string indicator (%variable%), batch files, 92
string search, FIND, 297-298
SUBST, 85-86, 116, 220, 390-391
  ASSIGN use, 221, 391
  BACKUP use, 228, 391
  CONFIG.SYS use, 391
  COPY use, 391
  DISKCOMP use, 263, 391
  DISKCOPY use, 391
  FORMAT use, 391
  JOIN use, 320
  LABEL use, 322
  LASTDRIVE use, 50, 323, 391
  MKDIR use, 336
  REPLACE use, 391
  RESTORE use, 391
  TRUENAME use, 397
  XCOPY use, 391
suppression (@) command, batch files, 92
SWITCHES, 6, 210, 392
  ANSI.SYS use, 392
  CONFIG.SYS use, 41, 51-52
  IO.SYS support, 318
syntax, 210
SYS, 116, 393
SYS files (*see* system files)
SYSINIT, 16
system files, 5, 9
  bootable-disk creation, SYS, 393
  hidden, IO.SYS, 318-319
  MSDOS.SYS, 347
  set/disable attributes, ATTRIB, 222-224

T

target disk, copying, 264
task switching, DOSSHELL, 180-183
technical support, xvi
terminal devices, change, CTTY, 247-248
TIME, 8, 22, 249, 394
  international setting, COUNTRY.SYS, 246
transmission rates, MODE, 340
TREE, 78, 119, 138-140, 395-396, 407
  DOSSHELL use, 395
  SHELL use, 396
TRUENAME, 8, 22, 397
TSR programs, load, INSTALL, 317
TYPE, 8, 22, 398-399
  DEBUG use, 399
  EDIT use, 398, 399
  MORE use, 345, 398, 399
  PRINT use, 355, 398

U

UNDELETE, xiii, xv, 116, 129, 134, 150, 153-157, 160, 173, 400-401
  DELETE use, 400
  ERASE use, 400
  FORMAT use, 302-303
  MIRROR use, 331-334, 400-401
UNERASE, DEL use, 254
UNFORMAT, xiii, xv, 37, 116, 134, 150, 153, 160-168, 173, 402-404
  FORMAT use, 302-303
  MIRROR use, 331-334, 403
  RECOVER use, 365
UNINSTALL programs, 30-31
UNIX, 4
upgrade considerations, xvii-xviii, 30-31

V

variables, 210
VER, 8, 22, 405
VERIFY, 8, 22, 86, 406
  CONFIG.SYS use, 41, 52
version numbers, VER, 3-5, 405
virtual disks, 238
VMAP optimizer, 23
VOL, 8, 22, 407
volume label
  create, LABEL, 322
  display, VOL, 407

W

Weitek coprocessor, on/off (*see also* EMM386), 284
wildcards
  BACKUP use, 225
  FIND command, 211, 297
  REPLACE, 147-148
  XCOPY, 147-148
Windows, 18, 66, 67, 182, 296

X

XCOPY, 7, 116, 132, 144, 145, 146, 147, 149, 211, 408-409
  BACKUP use, 408
  COPY use, 244-245, 408
  DISKCOMP use, 263-265
  DOSKEY macros use, 409
  DRIVER.SYS use, 275-276
  ERRORLEVEL use, 409
  file attributes, ATTRIB use, 223
  FOR use, 300
  HELP, 72
  REPLACE use, 370
  RESTORE use, 408
  SUBST use, 391
  wildcard use, 147-148
XDV.COM, 22
XENIX, 13
XMA2EMS.SYS, 110
XMAEM.SYS, 110
XTREE, 78, 174, 320

DOS 5 represents a significant change, not only in DOS itself, but also in Microsoft's philosophy—indeed, DOS 5 acknowledges the realities of today's personal computer world. Much more than just a warmed-over version of DOS 4.01, the new DOS has many powerful features never seen in DOS before, plus a number of important enhancements of familiar ones.

In ***DOS 5 Demystified,*** James Forney explores the strengths, weaknesses, and peculiarities of this new release, offering insider tips to help you get the most from DOS 5's improved features. He also emphasizes compatibility issues and productivity strategies to help you take full advantage to DOS 5.

Forney includes DOS commands, functions, and features—each with a version specification—and he explains how you can run DOS 5 in memory above 640K. (One of the major improvements in DOS 5 is support for expanded memory.)

Even if you're a DOS beginner, Forney's "tips, tricks, and traps" approach to MS-DOS is guaranteed to turn you into a power user with full command over your operating system.

IBM DOS

Beginner	Intermediate	Advanced
✓	✓	

Praise for James Forney's *MS-DOS Beyond 640K:*

"The book lays out the alternatives for DOS users who run into the 640K byte barrier. It explains the ins and outs of extended and expanded memory, DOS extenders and alternative operating systems like PC-MOS and Concurrent DOS."

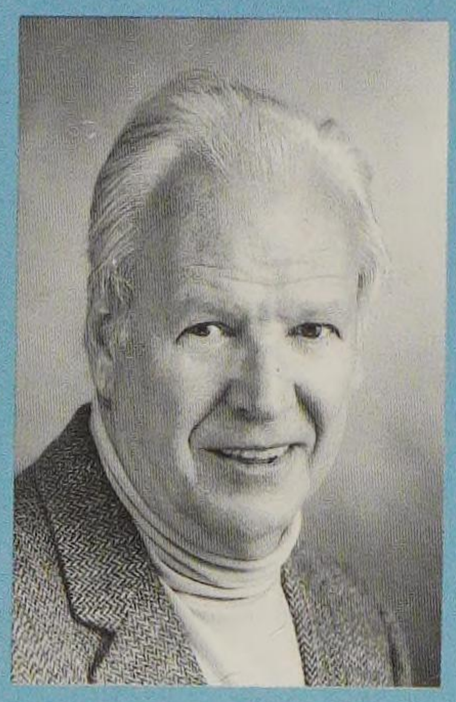
—Computer Reseller News

"Quite informative . . . If you want to clear up the confusion over extended and expanded memory and learn all about each, Forney's book should do it."

—David W. Batterson, Nationally Syndicated Book Reviewer

"For more information on basic memory concepts, read James Forney's MS-DOS Beyond 640K."

—David Essex, PC Resource Magazine



About the Author

James S. Forney is a contributing editor to the *Computer Buyer's Guide*, and his articles appear frequently in *PC Magazine*. Forney is the author of *The DOS Construction Kit* and the bestselling *MS-DOS Beyond 640K*.

0891 U.S.$24.95

ISBN 0-8306-1047-2

90000

9 780830 610471

Windcrest®/McGraw-Hill
Blue Ridge Summit, PA 17294-0850